

Generalization-Based Data Mining in Object-Oriented Databases Using an Object Cube Model ^{*}

Jiawei Han[§]

Shojiro Nishio[†]

Hiroyuki Kawano[‡]

Wei Wang[§]

[§] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6

[†] Department of Information Systems Engineering, Osaka University, Osaka 565, Japan

[‡] Department of Applied Systems Science, Kyoto University, Kyoto 606, Japan

{han@cs.sfu.ca, nishio@ise.osaka-u.ac.jp, kawano@kuamp.kyoto-u.ac.jp, weiw@cs.sfu.ca}

Abstract

Data mining is the discovery of knowledge and useful information from the large amounts of data stored in databases. With the increasing popularity of object-oriented database systems in advanced database applications, it is important to study the data mining methods for object-oriented databases because mining knowledge from such databases may improve understanding, organization, and utilization of the data stored there.

In this paper, issues on generalization-based data mining in object-oriented databases are investigated in three aspects: (1) generalization of complex objects, (2) class-based generalization, and (3) extraction of different kinds of rules. An object cube model is proposed for class-based generalization, on-line analytical processing, and data mining. The study shows that (i) a set of sophisticated generalization operators can be constructed for generalization of complex data objects, (ii) a dimension-based class generalization mechanism can be developed for object cube construction, and (iii) sophisticated rule formation methods can be developed for extraction of different kinds of knowledge from data, including characteristic rules, discriminant rules, association rules, and classification rules. Furthermore, the application of such discovered knowledge may substantially enhance the power and flexibility of browsing database, organizing database, and querying data and knowledge in object-oriented databases.

Keywords: Data mining, knowledge discovery in databases, object-oriented databases, object cube model.

^{*}This research was supported in part by the following: for the first author, the grants from the Natural Sciences and Engineering Research Council of Canada and NCE/IRIS; for the second author, the Ministry of Education, Science, Sports and Culture of Japan under Scientific Research Grant-in-Aid, for the first three authors, the Research Grant-in-Aid from the Artificial Intelligence Research Promotion Foundation, Japan, and for the third author, a scholarship from the Ministry of Education, Science and Culture of Japan and the work was done during his visit to Simon Fraser University.

1 Introduction

With the rapid growth in the amount of information stored in databases, the development of efficient and effective tools for *knowledge discovery in databases* (*KDD*, or *data mining*) has become an increasingly important task in database, statistics, machine learning, and data visualization researches [46, 17].

In the past several years, fruitful researches have been conducted on knowledge discovery in relational databases, with some experimental systems, such as Intelligent Miner (Quest) [3], MineSet [9], DBMiner [25], IMACS [8], SKICAT [16], Explora [36], etc. constructed and tested in large databases. Since object-oriented database systems [7, 10] are popular and influential in advanced database applications, it is important to extend our domain of study from relational database systems to object-oriented database systems and investigate the mechanisms for knowledge discovery in object-oriented databases (*OODBs*) [43, 29].

Object-oriented data models and systems [5, 6, 31, 33] embody rich data structures and semantics in the construction of complex databases, such as complex data objects, class/subclass hierarchies, class composition hierarchies, property inheritance, methods and active data, etc. This not only brings the power and flexibility to the system but also adds complexity to the implementations, including the development of knowledge discovery mechanisms [37].

In this article, issues on knowledge discovery in object-oriented database systems are investigated with some interesting generalization-based knowledge discovery mechanisms developed. To make the work generally applicable, the study does not assume a particular object-oriented model or system but is based on the generally recognized features of object-oriented database systems.

In our framework, a data mining process, initiated by a knowledge discovery query, which indicates the kind of rules to be discovered, the set of relevant data, and the background knowledge, proceeds as follows. First, a data retrieval process is initiated to collect the set of relevant data, which corresponds to the processing of an object-oriented database query (by some interesting object-oriented query processing techniques, such as [6, 32, 33, 51, 44]). Second, generalization is performed on the set of retrieved data, which corresponds to the generalization of complex data objects, for which a set of generalization operators should be developed. Third, an object cube-based generalization is performed to generalize and compress the set of relevant data into a compact generalized object cube. Finally, different kinds of knowledge rules can be extracted from the generalized object cube.

The paper is organized as follows. In Section 2, philosophical considerations and primitives for knowledge discovery in object-oriented databases are introduced. In Section 3, operators for generalization of complex data objects are examined. In Section 4, methods for object cube-based generalization are investigated. In Section 5, techniques for extraction of different kinds of rules are studied. In Section 6, other methods for mining in object-oriented databases and the applications of discovered knowledge are discussed, and the study is summarized in Section 7.

2 Primitives for knowledge discovery in object-oriented databases

There are different philosophical considerations on knowledge discovery in databases [21, 20, 58], which may lead to different methodologies in the development of knowledge discovery techniques. To simplify our discussion, an evolutionary approach is adopted, that is, starting from simple assumptions and relaxing them step-by-step to handle more complicated cases.

Assumption 2.1 *An object-oriented database stores a large amount of relatively reliable and stable data.*

Data in an object-oriented database may be incomplete, redundant, incorrect, or highly dynamic in certain applications [58], which makes knowledge discovery a challenging task. The assumption on data stability and reliability facilitates first developing knowledge discovery mechanisms in relatively simple environments and then evolving the techniques step-by-step towards more complicated situations.

Assumption 2.2 *A knowledge discovery process is initiated by a user's knowledge discovery query.*

Idealistically, one may expect that a knowledge discovery system will perform interesting discovery autonomously without human interaction. However, since generalization and knowledge extraction can be performed in many different ways on any subset of data in the database, huge amounts of knowledge may be generated from even a medium-sized database by unguided, autonomous discovery, whereas much of the discovered knowledge could be out of user's interest. In contrast, a command-driven and exploratory discovery may lead to guided and interactive discovery with a focus on the interested set of data and therefore represent constrained search for the desired knowledge. Thus, command-driven and exploratory discovery is adopted in this study.

Assumption 2.3 *Generalized rules are expressed in terms of high level concepts.*

Without concept generalization, discovered knowledge is expressed in terms of low-level data (data stored in the databases), often in the form of functional dependencies, low-level association rules [4], or low-level integrity constraints. On the other hand, with concept generalization, discovered knowledge can be expressed in terms of concise, expressive and higher level abstraction, in the form of generalized rules or generalized constraints, and be associated with statistical information. It is often desirable for large databases to have rules expressed at concept levels higher than the primitive ones.

Assumption 2.4 *Background knowledge is generally available for knowledge discovery process.*

Discovery may be performed either with the assistance of relatively strong background knowledge (such as conceptual hierarchy information, etc.) or with little support of background knowledge. Obviously, the discovery of conceptual hierarchy information itself can be treated as a part of knowledge discovery process. However, the availability of relatively strong background knowledge not only improves the efficiency of the discovery process but also expresses user's preference for guided generalization, which may lead to an efficient and desirable generalization process.

Following these assumptions, three primitives should be provided in a knowledge discovery process: the set of data in relevance to a discovery process, the kind of knowledge to be discovered, and the background knowledge.

The first primitive, the set of relevant data, can be specified in a way similar to that of an object-oriented query, which is to be used to fetch the set of relevant data from the database.

The second primitive, the kind of knowledge to be discovered, may include characteristic rules, discriminant rules, classification rules, association rules, etc. A **characteristic rule** is an assertion which characterizes a concept satisfied by all or most of the examples in the class undergoing examination (called the **target class**). For example, the symptoms of a specific disease can be summarized by a characteristic rule. A **discriminant rule** is an assertion which discriminates a concept of the class being examined (the **target class**) from other classes (called **contrasting classes**). For example, to distinguish one disease from others, a discriminant rule should summarize the symptoms that discriminate this disease from others. A **classification rule** first returns a preferred classification scheme for a discovery request and then a set of rules associated with each class or subclass. For example, one may classify diseases and provide the symptoms which describe each class or subclass. An **association rule** describes association relationships among a set of data (patterns). For example, one may discover a set of symptoms frequently occurring together with certain kinds of diseases and further study the reasons behind it.

The third primitive, the background knowledge, is the concept hierarchies or generalization operators which provide corresponding higher level concepts and assist generalization processes, which will be discussed in detail in the next section.

Based on these primitives, a knowledge discovery language for object-oriented databases, OML (Object-oriented knowledge Mining Language), is proposed for high level knowledge discovery query specification, with a syntax similar to SQL.

In general, a knowledge discovery task is specified in the following extended BNF, where “[]” represents 0 or one occurrence, “{ }” represents 0 or more occurrences, and words in **sans serif** font represent keywords. Notice that `rule_spec` is the specification of the kind of rules to be discovered, which are in different formats for discovering different kinds of rules.

```
use database_name
{use hierarchy for attribute(s)}
rule_spec
from class(es)
where condition
with respect to attribute(s)
{with [kinds_of] threshold = threshold_value [for attribute(s)]}
```

Different knowledge discovery tasks have different rule specifications to fit in the `rule_spec` statement. The `rule_spec` for generalization of a set of data and for the discovery of characteristic rules, discriminant rules, association rules, and classification rules are presented as follows.

1. The specification for data generalization is as follows.

rule_spec ::= generalize data [into “class_name”]

2. The specification for the discovery of characteristic rules is as follows.

rule_spec ::= mine characteristic rules [as “rule_name”]

3. The specification for the discovery of discriminant rules is as follows.

rule_spec ::= mine discriminant rules [as “rule_name”]
 for class_1 where condition_1
 in contrast to class_2 where condition_2
 {in contrast to class_i where condition_i}

4. The specification for the discovery of classification rules is as follows.

rule_spec ::= mine classification [according to class attribute class_attribute(s)]

5. The specification for the discovery of association rules is as follows.

rule_spec ::= mine association rules [as “rule_name”]

Concrete examples of these knowledge discovery queries and their implementations will be examined in detail in the following sections.

3 Operators for Generalization of Attributes, Methods, and Complex Object Components

An object-oriented database organizes a large set of complex data objects into classes which are in turn organized into class/subclass hierarchies. Each object in a class is associated with (1) an object-identifier, (2) a set of attributes which may contain sophisticated data structures, set- or list- valued data, class composition hierarchies, multimedia data, etc. and (3) a set of methods which specify the computational routines or rules associated with the object class.

To facilitate the development of knowledge discovery mechanisms in object-oriented databases, it is important to implement efficiently a relatively small set of generalization operators on which a large set of possible generalization operations can be developed.

Formally, the generalization of a component p of an object O_i can be written in an abstract way as $Gen(O_i.p)$, where Gen is an abstract object generalization operator which can be transformed into a concrete operation based on the role of the component and the specific learning requirement. Moreover, the generalized component of an object can be further generalized by applying Gen again, which could be the same or different generalization operators compared with the one applied in the last generalization. If the generalization is performed by applying the same sequence of generalization operators on a component p of O_i , we should have

$$Gen^{n-1}(Gen(O_i.p)) \equiv Gen(Gen^{n-1}(O_i.p)).$$

For example, a person P_1 's address can be generalized from a detailed address, such as the number of a street, into a higher leveled one, such as a street block, then a street, a district, a city, a province, a country, etc. when necessary by applying the generalization operator Gen several times, such as $Gen(Gen(\dots Gen(P_1.address) \dots))$.

An object in an object-oriented database is described by a set of attributes and a set of methods. The attribute value of an object could be a character, a fixed length character string, a numerical value, a structure, a class composition hierarchy, a set-valued or list-valued data, or unformatted data, such as text, map, image, voice, or other forms of multimedia data. The generalization of different and complex kinds of attribute values is a challenging task.

Suppose that an object property (attribute value) of an object O_i is p_k , its minimum generalized concept is p_{k-1} , which in turn has the minimum generalized concept p_{k-2} , etc. A sequence of generalization on the object property p_k can be represented as:

$$\forall j(1 \leq j \leq k) \quad Gen(p_j) = p_{j-1}. \quad (3.1)$$

In this section, we examine the generalization of different components of complex objects in an object-oriented database, including object identifiers, unstructured and structured values, class composition hierarchies, spatial and multimedia data, inherited and derived data, methods specified by rules or procedures, etc.

3.1 Generalization of object identifiers and class/subclass hierarchies

One of the essential components of an object-oriented database is *object identifier* which uniquely identifies objects. It remains unchanged over structural reorganization of data. At the first glance, it seems to be impossible to generalize an object identifier. However, since objects in an object-oriented database are organized into classes which in turn belong to a class/subclass hierarchy, the generalization of objects may refer to their corresponding class/subclass hierarchy. Therefore, an object identifier can be first generalized to the corresponding lowest subclass name which can in turn be generalized to a higher level class/subclass name by climbing up the class/subclass hierarchy. For the same reason, a class or a subclass can be generalized to its corresponding superclass(es) by climbing up the class/subclass hierarchy.

Suppose the object identifier of an object O_i is $O_i.oid$, the name of the lowest subclass that O_i belongs to is $C_{i,j}$, and the name of the superclass of a class $C_{i,k}$ is $C_{i,k-1}$, for all k ($1 \leq k \leq j$). The generalization on the object identifier $O_i.oid$ for O_i and its corresponding classes can be represented as:

$$Gen(O_i.oid) = C_{i,j}. \quad (3.2)$$

$$\forall k(1 \leq k \leq j) \quad Gen(C_{i,k}) = C_{i,k-1}. \quad (3.3)$$

3.2 Generalization on single attribute values

Single valued, numerical, and nonnumerical data are the most frequently encountered attribute values in databases.

Generalization on nonnumerical values may rely on the available concept hierarchies specified by domain experts or users. Concept hierarchies represent necessary background knowledge which directs the generalization process. Different levels of concepts are often organized into a taxonomy or hierarchy of concepts which can be partially ordered according to a general-to-specific ordering. The most general concept (corresponding to *level 0*) is the null description (described by a reserved word “any”), and the most specific concepts correspond to the specific values of attributes in the database. Using a concept hierarchy or taxonomy, the discovered rules can be represented in terms of generalized concepts and stated in a simple and explicit form, which is desirable to most users.

Many concept hierarchies, such as *geo_location(street, city, province, country)*, are stored in the database implicitly, which can be made explicit by specifying certain attribute generalization meta-rules, such as generalization by removing attributes *street* and *city* in *geo_location*, etc. Some hierarchies can be provided by knowledge engineers or domain experts, which is reasonable even for large databases since a concept hierarchy registers only the *distinct* discrete attribute values or ranges of numerical values for an attribute which are, in general, not very large. Some hierarchy could be derived from the knowledge stored elsewhere, or being computed by applying some rules or algorithms, such as deriving “*British Columbia* \Rightarrow *Western Canada*” from a geographic map stored in a spatial database, etc. Also, a hierarchy could be defined on a single attribute or on a set of related attributes, and it could be in the shape of a balanced tree, an unbalanced tree, a lattice or a DAG (directed acyclic graph).

To certain extent, concept hierarchies can also be generated automatically [18, 14] based on user’s learning requirements, data semantics and/or data distribution statistics. Moreover, a given concept hierarchy may not be best suited for a particular learning task, which therefore often needs to be dynamically refined based on data distribution statistics for desired learning results [26]. For example, if the learning requirement is to analyze the birth place of the *students* of Simon Fraser University, the level 1 (top-level) concepts could be: $\{B.C, other_provinces_in_Canada, foreign\}$. However, if it is to analyze the birth place of the *faculty* of the university, the appropriate level 1 concepts could be $\{North_America, Europe, Asia, other_countries\}$. Both concept hierarchies can be obtained by dynamic adjustment of a given concept hierarchy based on the analysis of the statistical distribution of the relevant data sets.

Different concept hierarchies can be constructed on the same attribute based on different viewpoints or preferences. For example, the birthplace can be organized according to administrative regions, geographic locations, size of cities, etc. Usually, a popularly referenced hierarchy is associated with an attribute as the default one. Other hierarchies can be chosen explicitly by users in a learning process. A concept hierarchy can also be in the shape of lattice or DAG. Moreover, it is sometimes preferable to perform induction in parallel along with more than one hierarchy and select a desired one based on some intermediate generalization results.

Generalization on numerical attributes can be performed similarly but in a more automatic way by the examination of data distribution characteristics [18]. A predefined concept hierarchy may not be required in many cases. For example, the ages of the graduate students in a university can be clustered into several groups, such as $\{below\ 23, 23-26, 26-30, over\ 30\}$, according to a relatively uniform data distribution criteria or using some statistical clustering analysis tools. Appropriate names can be assigned to the generalized numerical ranges, such as $\{very\ young, young, \dots\}$ by users or experts to convey more semantic meaning.

3.3 Generalization on structured data

Complex structure-valued data, such as set-valued and list-valued data and data with nested structures, are common in object-oriented databases. They can be generalized in several ways in order to extract interesting patterns from such data sets.

A set-valued attribute may be of homogeneous or heterogeneous types. Typically, a set-valued data can be generalized in two ways: (1) generalization of each value in a set into its corresponding higher level concepts, or (2) derivation of the general behavior of a set, such as the number of elements in the set, the types or value ranges in the set, the weighted average for numerical data, etc. Notice that in the case of set-valued attribute generalization, the generalization operator $Gen(p_k)$ indicates that the input p_k can be a set of values and the output " $p_{k-1} = Gen(p_k)$ " may also be a set of values. Moreover, generalization can be performed by applying different generalization operators to explore alternative generalization paths. In this case, the result of generalization is a heterogeneous set.

For example, the *hobby* of a person is a set-valued attribute which contains a set of values, such as $\{tennis, hockey, chess, violin, nintendo\}$, which can be generalized into a set of high level concepts, such as $\{sports, music, video_games\}$, or into 5 (the number of hobbies in the set), or both, etc. Moreover, a *count* can be associated with a generalized value to indicate how many elements are generalized to the corresponding generalized value, such as $\{sports(3), music(1), video_games(1)\}$, where $sports(3)$ indicates *three kinds of sports*, etc.

A set-valued attribute may be generalized into a set-valued or a single-valued attribute; whereas a single-valued attribute may also be generalized into a set-valued one if the "hierarchy" is a lattice or the generalization follows different paths. Further generalizations on such a generalized set-valued attribute should follow the generalization path of each value in the set.

A list-valued or a sequence-valued attribute can be generalized in a way similar to the set-valued attribute except that the order of the elements in the sequence should be observed in the generalization. Each value in the list can be generalized into its corresponding higher level concept. Alternatively, a list can be generalized according to its general behavior, such as the length of the list, the type of list elements, the value range, weighted average value for numerical data, or dropping unimportant elements in the list, etc. A list may be generalized into a list, a set or a single value. For example, a sequence (list) of data for a person's education record: " $((B.Sc. in Electrical Engineering, U.B.C., Dec., 1980), (M.Sc. in Computer Engineering, U. Maryland, May, 1983), (Ph.D. in Computer Science, UCLA, Aug., 1987))$ " can be generalized by dropping less important descriptions (sub-attributes) of each tuple in the list, such as " $((B.Sc., U.B.C., 1980), \dots)$ ", or by retaining only the most important tuple(s) in the list, or both, such as " $(Ph.D. in Computer Science, UCLA, 1987)$ ".

Set- and list-valued attributes are simple structure-valued attributes. In general, a structure-valued attribute may contain sets, tuples, lists, trees, records, etc. and their combinations. Furthermore, one structure can be nested in another structure at any level. Similar to the generalization of set- and list-valued attributes, a general structure-valued attribute can be generalized in several ways, such as (1) generalizing each attribute in the structure whereas maintaining the shape of the structure, (2) flattening the structure and generalizing on the flattened structure, (3) removing the low-level structures or summarizing the low-level structures by high-level concepts or aggregation, and (4) returning the type or an overview of the structure.

3.4 Aggregation and approximation as a means of generalization

Besides concept tree ascension and structured data summarization, aggregation and approximation should be considered as an important means of generalization, which is especially useful for generalizing the attributes with large sets of values, complex structures, spatial or multimedia data, etc.

Take spatial data as an example. It is desirable to generalize detailed geographic points into clustered regions, such as business, residential, industry, or agricultural areas, according to the land usage. Such generalization often requires the merge of a set of geographic areas by spatial operations, such as spatial union or spatial clustering algorithms. Aggregation and approximation are important techniques in such generalization. In spatial merge, it is necessary not only to merge the regions of similar types within the same general class but also to compute the total areas, average density or other aggregate functions and ignore some scattered regions with different types if they are unimportant to the study. For example, different pieces of land for different purposes of agricultural usage, such as vegetables, grains, fruits, etc. can be merged into one large piece of land by spatial merge. However, such an agricultural land may contain highways, houses, small stores, etc. If the majority land is used for agriculture, the scattered spots for other purposes can be ignored, and the whole region can be claimed as an agricultural area by approximation. The spatial operators,

such as *spatial_union*, *spatial_overlapping*, *spatial_intersection*, etc., which merge scattered small regions into large, clustered regions can be considered as generalization operators in spatial aggregation and approximation.

3.5 Generalization on multimedia data

A multimedia database may contain complex texts, graphics, images, maps, voice, music, and other forms of audio/video information. Such multimedia data are typically stored as sequences of bytes with variable lengths, and segments of data are linked together for easy reference. Generalization on multimedia data can be performed by recognition and extraction of the essential features and/or general patterns of such data.

There are many ways to extract the essential features or general patterns from segments of multimedia data. For an image, the size and color of the contained objects or the major regions in the image can be extracted by aggregation and/or approximation. For a segment of music, its melody can be summarized based on the approximate patterns that repeatedly occur in the segment and its style can be summarized based on its tone, tempo, major musical instruments played, etc. For an article, its abstract or general organization such as the table of contents, the subject and index terms frequently occurring in the article, etc. may serve as generalization results. In general, it is a challenging task to generalize multimedia data to extract the interesting knowledge implicitly stored in the data. Further research should be devoted to this issue.

3.6 Generalization on inherited and derived properties

Object-oriented databases are organized into class/subclass hierarchies. Some attributes or methods of an object class are not explicitly specified in the class itself but are inherited from its higher level classes. Some object-oriented database systems may allow the properties to be inherited from more than one superclass (called *multiple inheritance*) when the class/subclass “hierarchy” is organized in the shape of a lattice. The inherited properties of an object can be derived by query processing in the object-oriented database. From the knowledge discovery point of view, it is unnecessary to distinguish which data are stored within the class and which are inherited from its superclass. As long as the set of relevant data are collected by query processing, the knowledge discovery process will treat the inherited data in the same way as the data stored in the object class and perform generalization accordingly.

Method is another important component of object-oriented databases. Many behavioral data of objects can be derived by application of methods. Since a method is usually defined by a computational procedure/function or by a set of deduction rules, it is difficult to perform generalization on the method itself unless the generalization of the method is clearly understood by application programmers and is coded as a new method which directly performs the required generalization. In general, the generalization on the data derived by method application should be performed in two steps: (1) deriving the task-relevant set of data by application of the method and, possibly, also data retrieval; and (2) performing generalization by treating the derived data as the existing ones.

3.7 Generalization on class composition hierarchies

An attribute of an object may be composed of or described by another object, and some of whose attributes may be in turn composed of or described by other objects, thus forming a class composition hierarchy. Generalization on a class composition hierarchy can be viewed as generalization on a set of (possibly infinite, if the nesting is recursive) nested structured data.

In principle, the reference to a composite object may traverse via a long sequence of references along the corresponding class composition hierarchy. However, in most cases, the longer sequence of references is traversed, the weaker semantic linkage between the original object and the referenced composite object is. For example, one attribute “*vehicles_owned*” of an object class “*student*” could refer to another object class “*car*” which may contain an attribute “*auto_dealer*”, which may refer to its “*manager*” with an attribute “*children*”. Obviously, it is unlikely to find any interesting general regularities between a student and his/her car’s dealer’s manager’s children.

Therefore, generalization on a class of objects should be mainly performed on its own immediate descriptive attribute values, methods, etc. with limited references to (and therefore generalization on) the indirectly referenced portion in the class composition hierarchy. That is, in order to discover relatively interesting knowledge, generalization should be performed only on the objects in the class composition hierarchy closely related in semantics to the currently focused class(es) but not on those which have only remote and rather weak semantic linkages.

4 Class Generalization

With the availability of operators for object generalization, one can explore the class-based generalization of a set of relevant objects. Since a set of objects in a class may share many attributes and methods, and the generalization of each attribute and method may apply a sequence of generalization operators as described in the previous section, the major issue becomes how to cooperate the generalization processes among different attributes and methods in the class(es) to produce interesting results.

Generalization of a class of objects can be viewed as a sequence of set-oriented generalization processes, each transforming a class into a relatively more generalized class. The class fed into a generalization process is called the *working class*, \mathcal{W} , with the initial one (i.e., the set of task-relevant data) called the *initial working class*, \mathcal{W}_0 . The class generated by the application of a generalization operator is called the *resulting class*, \mathcal{R} .

Different from most *learning-from-examples* algorithms [38, 22] which partition the set of examples into *positive* and *negative* sets and perform *generalization* using the positive data and *specialization* using the negative ones [38], a database usually does not explicitly store negative data, and thus no explicitly specified negative examples can be used for specialization. Therefore, a database induction process relies mainly on generalization, which should be performed cautiously to avoid over-generalization.

An attribute-oriented induction method, developed in the study of knowledge discovery in relational databases [24, 28], can be extended to object-oriented databases, as outlined below. Notice that in the following discussion, an attribute of a class may also represent that resulted from the application of a class-associated method. For example, “age” can be viewed as an attribute obtained by application of a method “*compute_age*” based on the current data and the value in the attribute “*birth_date*” of an object “*person*”. For class-based generalization, a set of generalization operators are selected and applied to an attribute in the working class without considering the inter-relationships among different attributes before the concepts in each attribute are generalized to a reasonably high level. The reason to ignore the examination of relationships among different attributes at an early stage of generalization is that such relationships, if considered at an early stage, would have to be expressed at an undesirably low level in large numbers. This cannot lead to elegant generalized rules to be expressed at a high level and in concise terms. Attribute-oriented induction, which considers the relationships among attributes only when the data have been generalized to relatively high concept levels, will substantially reduce the computational complexity of a database generalization process.

Formally, a generalization operator Gen can be applied to an attribute a_i on every object in a working class \mathcal{W}_k , resulting in a new generalized class \mathcal{R}_k . Thus, a **class generalization operator**, $ClassGen$, is introduced which applies the object generalization operator Gen to an attribute a_i of *every* object in the working class. That is,

$$\mathcal{R}_k = ClassGen(\mathcal{W}_k, a_i) = \{o' : (o \in \mathcal{W}_k) \wedge (o'.a_i = Gen(o.a_i)) \wedge (\forall(j \neq i) o'.a_j = o.a_j)\} \quad (4.4)$$

For example, for a working class “ $\mathcal{W}_0 = Person$ ” in a university database, $ClassGen(\mathcal{W}_0, address)$ derives a resulting class \mathcal{R}_0 with one-step generalization on the attribute “*address*” (e.g., from street number to street block) and all the other attributes unaltered.

A knowledge discovery process can be viewed as the application of a sequence of database generalization operators, $ClassGen$, on different attributes until the resulting class contains a small number of generalized objects which can be summarized as a concise, generalized rule in high-level terms.

4.1 An object cube model

An object-based attribute-oriented induction techniques has been developed in our previous studies of knowledge discovery in object-oriented databases [43, 29], whose general idea is outlined above.

Here we introduce an *object-based data cube* (or briefly **object cube**) model which provides a flexible induction technique for mining knowledge in object-oriented databases. Then we study the class generalization in object cube model.

A popular conceptual model that influence the design of data warehouses and development of query engines and front-end tools for on-line analytical processing (OLAP) is the *multidimensional model* of data in the data warehouse [12, 23]. A **multidimensional database**, also popularly called **data cube**, is a database consisting of a huge set of *facts* (also called *measures*) or multidimensional points and a relatively small set of *dimensions* with respect to which data is analyzed. For example, a *sales* data warehouse may store *sales amount* as *fact* in the fact table and take *store*, *product*, and *date* as *dimensions* and store their descriptions in the associated dimension tables.

The dimensional tables are where the textual description of the dimensions of the business are stored. Each dimension has a set of attributes. For example, a location dimension of a store may have attributes *number*, *street*, *city*, *province*, and *country*. Attributes of a dimension may be related by partial order, which may form either a hierarchy, such as “street → city → province → country”, which indicates that the data in the attributes form a hierarchical relationship, or a lattice, such as “day → month → quarter → year,” or “day → week → year”, which indicates that the attributes, “*day*, *week*, *month*, *year*”, form a lattice structure.

The hierarchy and lattice structures of dimension attributes are shown in Figure 1.

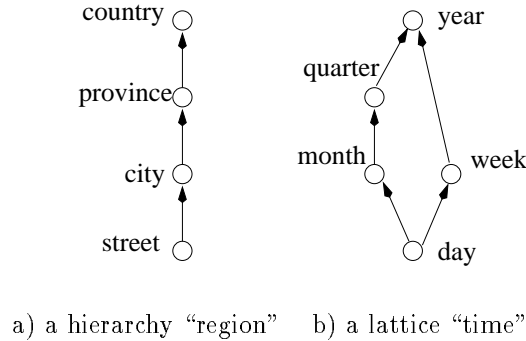


Figure 1: Hierarchical and lattice structures of attributes in warehouse dimensions.

The facts are *numerical measures* collected in the corresponding dimensions and provided in a data warehouse for data analysis, e.g., sales amount, budget, etc. The fact table stores the keys of multiple dimensions and the numerical measures of the business.

The object cube model is the further development of the data cube model for data warehousing in object-oriented databases.

Based on our discussion of object generalization methods in Section 3, an object-oriented database contains complex structured objects, spatial and multimedia data, class/subclass hierarchies, methods, etc. These complex structures can be generalized to simple and similar structures, and then the generalized database may share a good set of similar structures as that in relational databases. To that extent, a multidimensional database can be constructed on the generalized, object-oriented data in a way similar to that on relational data, and thus the object cube model can be built.

Definition 4.1 An **object cube** is a multidimensional database constructed on top of a generalized class in an object-oriented database. The generalized class contains a set of generalized attributes, serving as dimensions of the object cube, and one or a set of attributes, collecting aggregate values or other extents and serving as measures of the cube.

This definition indicates that a multidimensional database can be constructed if the object class resulted from some generalization of a portion of an object-oriented database forms a set of dimensions and measures in a way similar to that of relational databases. Notice that the measures of the multidimensional database may contain numerical aggregate values as well as *other extents* as measures where the *other extents* could be a collection of object identifiers or a collection of other features of generalized objects, which is explained as follows.

With the object generalization techniques discussed in Section 3, it is likely that many diverse features of complex objects can be generalized to sharable features which may form sharable dimensions in a multidimensional database. However, some features of a class of generalized objects may not be able to form sharable dimensions of a multidimensional database. Such features cannot contribute to the dimension formation in the object cube construction. They should either be further generalized to form new dimensions, or be excluded from the contents of the object cube, or be collected as a collection of object identifiers or object features to contribute to the construction of the extent measure of an object cube.

For example, a class of spatial objects may contain a diverse collection of shapes of spatial regions. The *shape* feature could be too diverse (e.g., almost every object has a distinct shape) to form a sharable dimension which usually contains a small number of distinct values. The feature *shape* should either be further generalized to form a smaller number of distinct but generalized shapes, such as *polygon*, *ellipse*, *circle*, *collection of polygons*, etc., or be excluded

from the contribution to the object cube construction (i.e., the constructed object cube will not contain the feature *shape*), or be collected as a collection of object pointers or distinct shape features such that a collection of similar generalized objects which share the same other generalized features will be “collapsed” into one cube cell with an extent measure containing a pointer pointing to a set of object identifiers or a set of distinct shapes.

From a philosophical point of view, one may consider that an object cube starts with a collection of all the relevant objects, which is a cube containing zero dimension and one extent measure where the extent is the collection of all the object identifiers of the relevant objects. With the class-based object generalization techniques discussed above, sharable dimensions can be extracted step by step to form new dimensions, which split the cube cell step by step into a collect of cells corresponding to each combination of values from different dimensions. Such sharable features will also lead to the generation of new measures, such as count or sum, and reduce the “load” in the extent. This process leads to the construction of an object cube containing a rich set of dimensions and measures.

Since an object cube is a multidimensional database containing a set of dimensions and measures, it can be modeled by the *star*, *snowflake*, and *fact constellation* models [12] as that in relational multidimensional databases.

Based on the above discussion, the computation of an object cube can be performed as follows. First, a class of objects interested in the study is collected by object-oriented query processing, which could be specified by an object-oriented query or simply done by a collection of potentially interested data sets. Second, object generalization methods discussed in Section 3 are applied to the collected class, which generalizes the data collected to a *minimally generalized class* which fits a multidimensional data model, with generalized attributes correspond to dimensions, and values aggregated into the cube cells as measures based on the measure computation methods defined in the object cube. This minimally generalized class forms the *base cuboid* of the object cube. Further generalization on the base cuboid derives other cuboids, which could either be precomputed before a query is submitted to the system and stored for future use, or be performed on the fly to answer a particular mining query. The computed object cube, consisting of based cuboids and other partially materialized cuboids, form the base for on-line analytical processing (OLAP) and data mining in object-oriented databases.

4.2 A class generalization algorithm for object cube construction

The construction of an object cube and the derivation of a prime class (cuboid) for object cube can be implemented by application of a sequence of database generalization operators, each on a corresponding *working class* and resulting in a more general *resulting class*, which in turn becomes the working class in the next round of database generalization. However, such an iterative generalization on a sequence of resulting class(es) may involve many passes of searching through large databases.

An efficient class generalization algorithm presented here scans the set of relevant objects at most twice. The first scan collects the statistical distribution information for the attribute values and determines the set of generalized values that each attribute should be generalized to. The information will be used for the preparation of generalization. If the data set is huge, sampling (which may cost much less than a full scan) can be performed on a portion of the data set to achieve satisfactory results. After determining the concept level for the prime class, the second scan will generalize all the properties of each object in the initial working class into the concepts at the level of the prime class (by generalization pair replacement) and merge the identical dimension values on the fly (with the *count* incremented, other numerical measures computed, and the extent collected), which derives the prime class efficiently.

Notice that from the object cube point of view, the prime class derived could be either a base cuboid, or some more generalized cuboid in the object cube lattice, depending on a preset *attribute threshold*, which is the maximum number of distinct values of an attribute in the class generalization. The base cuboid can be viewed as a special case of prime class in which each dimension’s attribute threshold is the largest possible one which makes the dimension formation feasible. In practice, if the attribute threshold for a dimension is specified by a user or by an expert (as a default one), the generalization will lead to a prime class which is usually a nonbase cuboid. However, if none of the dimension has a prespecified attribute threshold, the base cuboid is generated using the algorithm and is stored for flexible mining on top of it.

This technique is outlined in the following dimension-based induction algorithm.

Algorithm 4.1 (Dimension-based generalization in object-oriented databases) *Generalization in an object-oriented database by dimension-based induction based on a user’s generalization request.*

Input. 1. A *DBQuery* (written in the object-oriented knowledge mining language OML defined in Section 2) related to the object-oriented database and the generalization task,

2. $Gen(a_i)$, a set of concept hierarchies or generalization operators on dimension a_i 's, and
3. T_i , a set of *dimension thresholds* for dimension a_i 's.

Output. A *prime (generalized) class* (i.e., the *prime cuboid of the object cube*) based on the generalization request.

Method. The dimension-based induction is performed in the following three steps.

1. **InitClass:** Execute *DBQuery* to derive the *initial working class*, \mathcal{W}_0 , i.e., collect the set of task-relevant data by an object-oriented database query based on the generalization request.
2. **BaseGen:** Generalize the *initial working class*, \mathcal{W}_0 , minimally to form a minimally generalized base cuboid $Cube_0$.¹
 - (a) Collect the distinct values for each attribute a_i in the initial working class \mathcal{W}_0 and count the number of distinct values. For numerical values, generate minimal intervals for registration of the number of intervals. Based on the number of distinct values/intervals in each dimension, determine the minimal generalized level for each dimension of the base cuboid.
 - (b) Taking a group of hierarchical attributes as one dimension and *count*, *sum*, etc. as measures, construct the base cuboid $Cube_0$. The aggregate layers of the cuboid can be computed efficiently using a typical cube construction algorithm, such as [56].
3. **PreGen:** Prepare for prime cube generation.
 - (a) Compute the desired level L_i for each dimension a_i based on its dimension threshold T_i (automatic hierarchy generation can be performed for numerical data if there is none, and dynamic hierarchy adjustment can be performed for both numerical and nonnumerical data, if desired [26]). Notice that a_i should be removed if there is a large set of distinct values in a_i in \mathcal{W}_0 but there is no generalization operator on a_i .
 - (b) Determine the mapping-pairs $\langle v, v' \rangle$ for each dimension a_i , where v is a distinct value of a_i , and v' is its corresponding generalized value at level L_i .
4. **PrimeGen:** Derive the *prime class*, $Cube_p$.
 - (a) Generalize each object o in the initial base cuboid $Cube_0$ into o' by replacing each v in o by v' based on the mapping-pairs $\langle v, v' \rangle$ derived at *PreGen*.
 - (b) Insert such o' 's into the prime class, in which objects with identical attribute values stored in the same slot with their *counts* and other measures, if any, accumulated. \square

The complexity of Algorithm 4.1 is analyzed as follows. Step 1 of the algorithm is essentially an object-oriented database query whose processing efficiency depends on a particular query processing algorithm. The following theorem is about the processing efficiency of Steps 2 to 4.

Theorem 4.1 *The worst-case time complexity of Steps 2 to 4 in Algorithm 4.1 is $O(n)$, if an m -dimensional cube structure is used to store the base and prime cuboids.*

Proof sketch. Let m' be the number of dimensions in the initial working class \mathcal{W}_0 . Step 2 scans n objects in \mathcal{W}_0 usually only once but occasionally a small number of times in the base cuboid computation (see [56]). Thus the complexity in the computation using a compressed cube structure is $O(n)$. Step 3 decides the level to be generalized by scanning the dimensions only which is linear to the size of dimension only. Step 4 performs the generalization, by scanning the base cuboid only once, each mapping the corresponding mapping pairs determined at Step 3, into the corresponding cube cell of the prime cuboid, and accumulate the aggregation values, which cost in total $O(n)$. Summing up the costs of Steps 2 to 4, we have the time complexity $O(n)$ stated in the theorem. \square

4.3 Dimension-based generalization: An example

Example 4.1 Let an object-oriented database consist of a set of classes *Person*, *Organization*, etc. and their associated subclasses. A portion of class/subclass hierarchy is shown in Fig. 2, where a subclass *senior_professor* is a subclass of both *senior_employee* and *professor* (i.e., *multiple inheritance*). A portion of the conceptual hierarchy

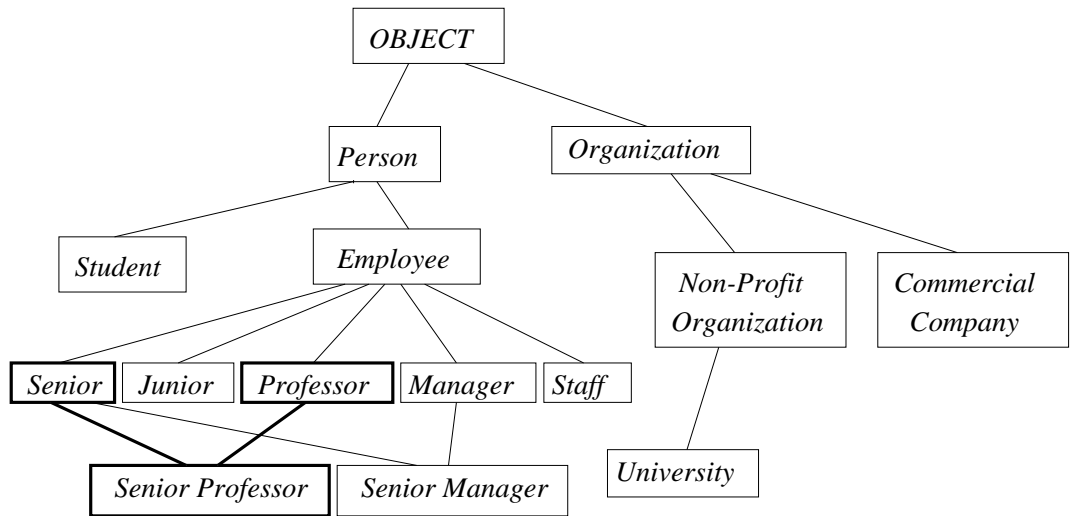


Figure 2: Class hierarchy in an object-oriented database.

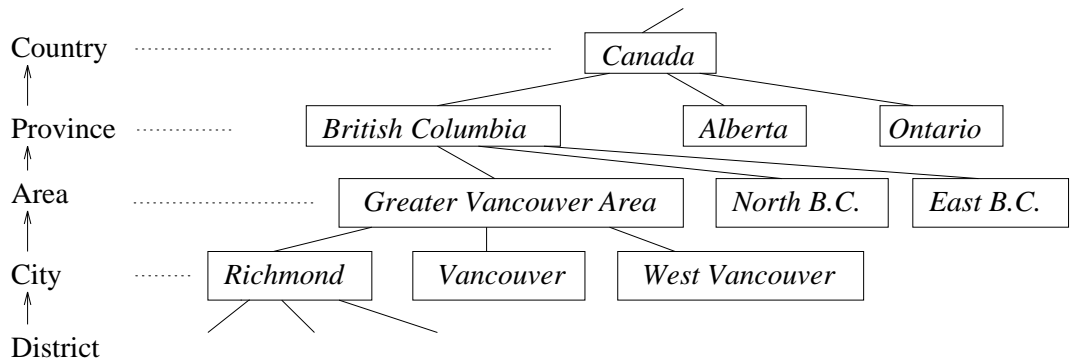


Figure 3: Concept hierarchy in an object-oriented database.

for the attribute *address* is shown in Fig. 3, where the concepts higher than *Canada* such as *North_America*, and that lower than the *city* level such as *district*, *street*, *block*, etc. are not presented in the figure.

Let the generalization task be to generalize a set of data in the class “**Person**”, in relevance to *house_size*, *residential area* and *salary*, for those with a Ph.D. degree, associated with U.B.C, 40 (years-old) or over, and driving Japanese cars.

The task can be represented in the following query [24] in the syntax of OML.

```

generalize Person P into Generalized_P
where P.Age >= 40 and P.Car.Maker = 'Japan'
      and 'Ph.D' in P.Education and P.Workplace.Name = 'U.B.C.'
in relevance to P.Name, P.Home.House_size, P.Salary, P.Home.Address
  
```

Following Algorithm 4.1, the class generalization request can be processed as shown below.

Step 1. Collect the set of objects that are related to the generalization task into the working class W_0 . One of the collected objects is supposed to be as follows. Notice that some attributes in the object could be defined by methods or be inherited from its superordinate classes. For example, “*age*” is a method which computes the age of a *Person* from the attribute *Birthdate* and the *current date*. Furthermore, the query processing will involve checking against “*Car_Object*”, “*Workplace_Object*” and “*Home_Object*” to satisfy the query condition.

Object[1]

¹The base cuboid *Cube₀* is used for drilling down when users would like to explore more details from the prime cube.

ObjectID: 02EREV ; String value is given by system
 Name: Alex Flemming ; String value
 Home: <House_ObjectID> ; House_Object has
 "House_size", "Address", "Price" and so on.
 Education: (..., (Ph.D. in Computer Science, UCLA, 1987)); Set-data values
 Workplace: <Workplace_ObjectID> ; Workplace_Object has
 "Name", "Address" and so on.
 Salary: \$73,854 ; Real value
 Birthdate: March 23, 1950 ; String value
 Age: Method(Birthdate, Today) ; Method
 Face: <Bitmap data> ; Multimedia data
 :

Step 2. In BaseGen, the minimally generalized base cuboid $Cube_0$ is generated which determines the dimensions to be generated based on the relevant set of attributes, and generalize the categorical data minimally and generate minimal intervals for numerical values. Taking a group of hierarchical attributes as one dimension and *count*, *sum*, etc. as measures, construct the base cuboid $Cube_0$. The compute the cube aggregation layers using a typical cube construction algorithm, such as [56].

Step 3. In PreGen, each dimension of the base cuboid $Cube_0$ is scanned once. The distributions of the dimension values are computed with the generalization mapping pairs determined. The attribute "P.Name" is removed since there are no superordinate concepts except "ANY" for a large number of distinct attribute values. Assuming that every attribute threshold is 4 by default. The generalization derives the following high level concepts associated with their corresponding low level concepts in the objects for the task-relevant attributes: $\{Richmond, Burnaby, Vancouver\}$ for *Address* (at the concept level of *city*), $\{senior_professor, senior_manager\}$ for object identifier (along the *class/subclass hierarchy*), $\{big, medium, small\}$ for *House_size*, and $\{L(low), M(medium), H(high)\}$ for *Salary*. Moreover, the total values of salaries at any generalized levels are also stored in the cells on the cube, we can easily compute the value of average salary from the *counts* and *total_salary* based on the object cube.

Step 4. In PrimeGen, objects are generalized based on the mapping pairs determined at Step 3, and each generalized object is inserted into the prime class. One of the generalized objects is shown below, where the *ObjectID* is a new object identifier created (and accessible only) by the system.

Generalized_Object[1]

Class: senior professor
 ObjectID: 60@WUH
 House_size: big
 Address: Richmond
 Salary: high
 Count: 103

If each generalized object is represented as a tuple, the prime class is represented as a relation as shown in Table 1, in which the OID attribute is omitted because it plays no role in the semantic interpretation of the generalized class. Moreover, a part of object cube with generalizing objects is also shown in Fig. 4, for example, the shaded cell in Fig. 4 has the *count* 103 and other attributes for the second tuple in Table 1.

<i>Class</i>	<i>House_size</i>	<i>Address</i>	<i>Salary</i>	<i>count</i>
<i>senior_professor</i>	<i>big</i>	<i>Richmond</i>	<i>medium</i>	64
<i>senior_professor</i>	<i>big</i>	<i>Richmond</i>	<i>high</i>	103
.....
<i>senior_manager</i>	<i>small</i>	<i>Vancouver</i>	<i>low</i>	49

Table 1: The prime class represented as a relation, which is a 4-dimensional cuboid.

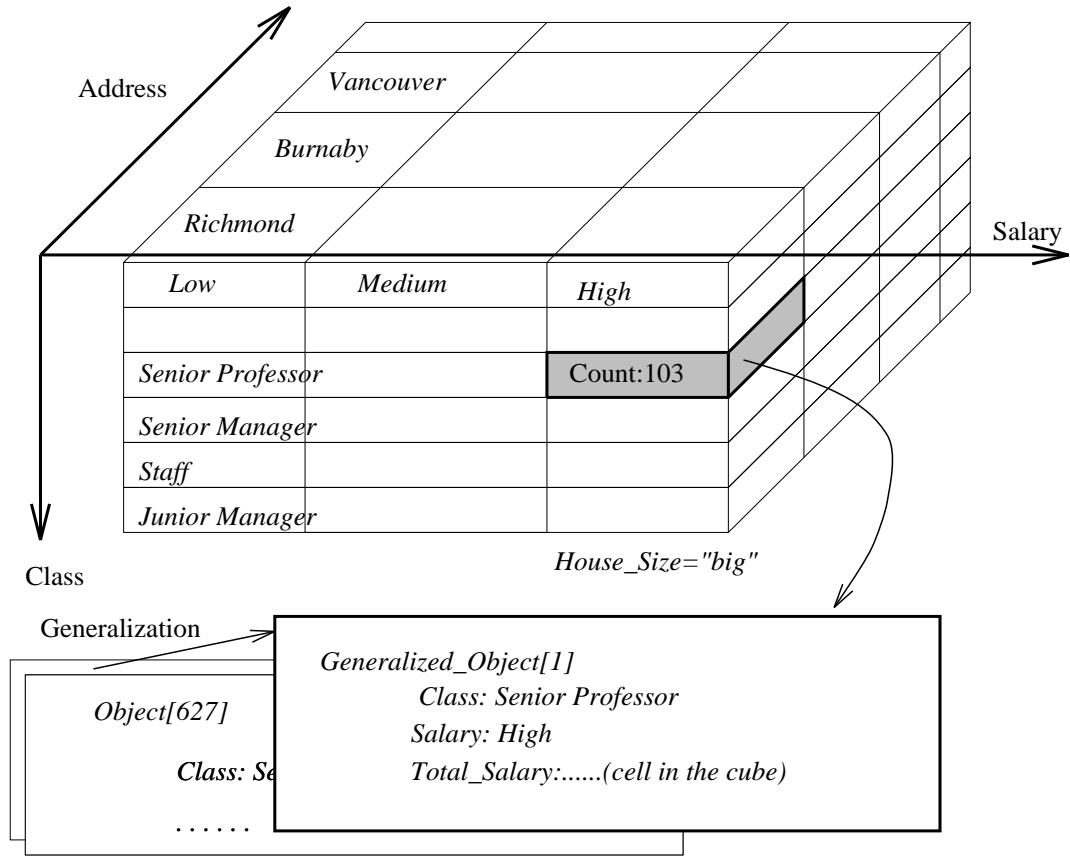


Figure 4: Multi-dimensional object cube with generalized objects.

The prime class can also be represented in the form of a multiple dimensional array, as shown by Table 2, which is called a *generalized cross-tabulation* or simply, *crossstab*, in which the first level columns show different subclasses: *senior professor* and *senior manager*, the second level columns show the salary level: *low*, *medium* and *high*, the first level rows show the house size: *big*, *medium* and *small*, and the second level rows show the address location: *Richmond*, *Burnaby* and *Vancouver*. □

5 Derivation of Generalized Rules

By performing dimension-based generalization, one or a set of prime (generalized) classes (cuboids) will be generalized from the relevant set(s) of data. Then different techniques can be applied on such prime generalized class(es) (cuboids) for extraction of different kinds of knowledge, concept classification, attribute reduction or further generalization, and compact rule generation. The generalized knowledge that can be derived includes characteristic rules, discriminant rules, classification rules, data evolution regularities, deviation rules, cluster description rules, and multiple-level association rules [28]. Moreover, knowledge can be represented in the form of generalized classes/relations, generalized crossstabs, or generalized rule(s).

This section studies the derivation of several kinds of knowledge from generalized data, including characteristic rules, discriminant rules, classification rules, and high-level association rules.

5.1 Extraction of characteristic rules

A **characteristic rule** is an assertion which characterizes the concepts satisfied by all or a majority number of the objects in a task-relevant set of data.

To extract interesting characteristic rules from a large set of data, it is expected that the concepts in the rules be represented at a relatively high level so that the rules can be represented concisely, summarizing the general behavior

		<i>senior_professor</i>				<i>senior_manager</i>				<i>Total</i>
		<i>L</i>	<i>M</i>	<i>H</i>	<i>Total</i>	<i>L</i>	<i>M</i>	<i>H</i>	<i>Total</i>	
<i>big</i>	<i>Richmond</i>	0	64	103	167	0	16	20	36	203
	<i>Burnaby</i>	2	27	46	75	1	8	9	18	93
	<i>Vancouver</i>	0	11	39	50	0	4	8	12	62
	<i>Total</i>	2	102	188	292	0	28	37	66	358
<i>medium</i>	<i>Richmond</i>	1	42	20	63	0	12	1	13	76
	<i>Burnaby</i>	0	21	62	83	0	3	18	21	104
	<i>Vancouver</i>	0	12	32	44	1	2	5	8	52
	<i>Total</i>	1	75	114	190	1	17	24	42	232
<i>small</i>	<i>Richmond</i>	4	10	0	14	8	0	0	8	22
	<i>Burnaby</i>	11	4	2	17	2	9	0	11	28
	<i>Vancouver</i>	35	24	5	64	49	41	6	96	160
	<i>Total</i>	50	38	7	95	59	50	6	115	210
<i>Total</i>		53	215	309	577	61	95	67	223	800

Table 2: The prime class (cuboid) represented as a generalized crosstab.

of the set of objects. Since the prime class is a generalized form of a set of relevant data, characteristic rules can be extracted from the prime class. Several techniques can be applied for such a knowledge extraction, including *direct presentation of the prime class as a prime (class) table*, *presentation of a prime class in the form of generalized crosstabs*, *further generalization of the prime class*, and *reduction of the number of attributes in the prime class*.

5.1.1 Prime class presentation: Prime table, crosstabs, and generalized rules

Direct presentation of a prime class in the form of a prime (class) table, such as Table 1 in Example 4.1, is the simplest and the most straightforward way of presenting a characteristic rule. Such a prime table can be viewed as a characteristic rule because it characterizes the general behavior of the set of data relevant to the mining task. Although it is in the form like a relation table, it is equivalent to a logic rule in the sense that the set of data which forms the class logically implies the general behavior of the class, represented by a set of generalized objects and their associated distributions. Such a presentation is effective if the prime class contains only a small number of generalized objects.

As an alternative representation, a prime class can be represented as a **generalized crosstab**, which can also be obtained by a mapping from the prime class table, by taking each attribute as one level of rows or columns in the table and each attribute value as one row or column at its corresponding level, with the occurrence count (and/or some other aggregated values, such as sum of the amount) of the corresponding objects in the class as the slot value, as demonstrated by Table 2 of Example 4.1.

Moreover, a generalized crosstab can be mapped into a set of smaller, **projected crosstabs**, each expressing the relationships among only a proper subset of the set of relevant attributes in the prime class. Such a representation is often desirable for examination of the general relationships among a subset of attributes. Different crosstabs can be extracted from a prime relation based on user's interest of different sets of reference attributes.

Furthermore, a prime class or a (projected) crosstab can be mapped into a logic rule with a *weight* associated with each disjunct to represent the *typicality* of each tuple of the prime class in the characteristic rule. Such a weight is called the *typicality-weight*, or briefly **t-weight**, defined as below.

Definition 5.1 Let q_a be a generalized tuple. The **t-weight** for q_a is the percentage of the original tuples covered by q_a in the target class. Formally, we have

$$t_weight = count(q_a) / \sum_{i=1}^n count(q_i),$$

where n is the number of tuples in the generalized relation, and q_a is in q_1, \dots, q_n .

Obviously, the range for t-weight is $[0 - 1]$. The rule can be represented either (i) in the form of a *weighted crosstab* by changing each *count* entry in the crosstab to the corresponding *t-weight* values, or (ii) in the logical form by associating a corresponding t-weight value with each disjunct.

Example 5.1 To extract weighted crosstabs for `House_size` related to `Class` and `Salary` respectively from the generalized class derived from Example 4.1, the following can be specified.

```
extract crosstabs according to House_size
from Generalized_P
with respect to Class, Salary
```

Tables 3 and 4 are two projected (weighted) crosstabs extracted from the prime crosstab, Table 2. Tables 3 represents the relationship between house size (`small`, `medium`, `large`) and categories (`senior professor`, `senior manager`) in the relevant set of data; whereas Tables 4 represents the relationship between house size (`small`, `medium`, `large`) and salary scales (`low`, `medium`, `high`) in the relevant set of data.

<i>House_size \ class%</i>	<i>senior_professor %</i>	<i>senior_manager%</i>	<i>Total%</i>
<i>Big</i>	36.5%	8.25%	44.75%
<i>Medium</i>	23.75%	5.25%	29%
<i>Small</i>	11.88%	14.37%	26.25%
<i>Total</i>	72.12%	27.88%	100.00%

Table 3: A crosstab for the relationship between house size and professional class.

<i>House_size \ salary%</i>	<i>Low%</i>	<i>Medium%</i>	<i>High%</i>	<i>Total%</i>
<i>Big</i>	0.38%	16.25%	28.12%	44.75%
<i>Medium</i>	0.25%	11.5%	17.25%	29%
<i>Small</i>	13.62%	11%	1.63%	26.25%
<i>Total</i>	14.25%	38.75%	47.00%	100.00%

Table 4: A crosstab for the relationship between house size and salary level.

The derivation of these projected (weighted) crosstabs can be easily performed by first performing projection on the prime class crosstab, Table 2, or direct extraction from the prime class, Table 1, and then replacing the entries with the corresponding t -weight values. \square

In general, a quantitative characteristic rule provides the necessary condition of the target class since the condition is derived based on all the facts in the target class, that is, the tuples in the target class must satisfy this condition. However, the rule may not be a sufficient condition of the target class since a tuple satisfying the same condition could belong to another class. Therefore, the rule should be in the form of

$$\forall x \text{ target_class}(x) \rightarrow \text{condition}_1(x)[t : w_1] \vee \dots \vee \text{condition}_n(x)[t : w_n].$$

The rule indicates that if x is in the *target_class*, there is a possibility of w_i that x satisfies *condition_i* where i is in $\{1, \dots, n\}$. Note that one may also drop some conditions if they are supported by very low t -weights, and the rules so generated will contain a smaller number of disjuncts and represent the summarization of a majority portion of the relevant data set.

Example 5.2 The crosstabs expressed by Tables 2, 3 and 4 can be viewed as characteristic rules since they represent certain general behavior of the relevant class of data and are logically equivalent to the corresponding characteristic rules. Such rules can be extracted by the following query.

```
mine characteristic rules
from Generalized_P
```

Let the class of data studied in Example 4.1 be **UBCJapCarDRs** (UBC Japanese car drivers). Equation (5.5) is the output of the query, a quantitative rule form of Table 4.

$$\begin{aligned}
 &UBCJapCarDrs(x) \rightarrow \\
 &\quad (house_size(x) = 'Big' \wedge salary(x) = 'Medium') [16.25\%] \\
 &\quad \vee (house_size(x) = 'Big' \wedge salary(x) = 'High') [28.12\%] \\
 &\quad \vee \dots\dots\dots \\
 &\quad \vee (house_size(x) = 'Small' \wedge salary(x) = 'High') [1.63\%] \tag{5.5}
 \end{aligned}$$

The rule implies that if x is in this class, there is 16.25% possibility that x earns medium salary and lives in big houses, 28.13% possibility that x earns high salary and lives in big houses, \dots , and only 1.63% possibility that x earns high salary and lives in small houses. \square

5.1.2 Further generalization and attribute reduction

When the prime class contains a relatively large set of generalized objects, techniques should be applied to reduce the size of the prime class in order to generate concise characteristic rules.

The generation of projected crosstabs shown in Example 5.1 is an effective method which projects the prime class on to the subsets of attributes, resulting in smaller projected crosstabs.

Another method to reduce the size of the prime class is to further generalize the prime class into a *reduced generalized class* in which the number of generalized objects is no more than a prespecified or default *class threshold* (i.e., the maximum number of objects that the generalized class may contain).

There are usually alternative choices to select a candidate attribute for further generalization. The interestingness of a derived generalized class relies on the selection of the attributes to be generalized and the selection of generalization operators, based on data semantics, user preference, generalization efficiency, etc. Criteria, such as the preference of a larger reduction ratio on the number of tuples or the number of distinct attribute values, the simplicity of the mined rules, etc., can also be used for selection.

Following different paths corresponds to the way in which different people may mine differently from the same set of examples. The generalized classes can be examined by users or experts interactively to filter out trivial rules and preserve interesting ones [58]. A graphical user interface may help users try different alternatives interactively in the process of mining desired results.

5.2 Extraction of discriminant rules

A **discriminant rule** is an assertion which discriminates concepts of the class being examined (the *target class*) from other classes (called *contrasting classes*). For example, to distinguish one disease from others, a discriminant rule should summarize the symptoms that discriminate this disease from others.

A discriminant rule can be discovered by generalizing the data in both target class and contrasting class(es) *synchronously* in an attributed-oriented fashion and excluding the properties that overlap in both classes in the generalized rule. Usually, a property, which is quite good at discriminating a class, may still have minor overlaps with other classes in a large data set. Thus it is often preferable to associate quantitative information with a rule or with the tuples in a generalized class to indicate how precisely a property can be used to distinguish a target class from others. Such a quantitative measurement, called *discriminating weight* [24], can be defined as the ratio of the frequency that a property occurring in the target class versus that occurring in both target and contrasting classes.

Formally, the *discriminating-weight*, or briefly *d-weight*, is defined as follows.

Definition 5.2 *Let q_a be a generalized concept (tuple) and C_j be the target class. The **d-weight** for q_a (referring to the target class) is the ratio of the number of original tuples in the target class covered by q_a to the total number of tuples in both the target class and the contrasting classes covered by q_a . Formally, the d-weight of the concept q_a in class C_j is defined as below.*

$$d_weight = count(q_a \in C_j) / \sum_{i=1}^k count(q_a \in C_i),$$

where k stands for the total number of the target and contrasting classes, and C_j is in $\{C_1, \dots, C_k\}$.

The range for d-weight is $[0 - 1]$. Obviously, a property with a discriminating weight close to 1 is a good discriminator; whereas that close to 0 is a negative discriminator (the properties unlikely occurring in the target class).

By associating d-weights, a discriminant rule provides a quantitative criterion to describe the general properties which can be used to distinguish a target class from the contrasting class(es). A quantitative discriminant rule is represented by a set of tuples in the target class, which can be represented (i) in the crosstab form by changing the *count* entry to the corresponding d-weight values, or (ii) in the logical form with a d-weight associated with each disjunct.

Logically, a quantitative discriminant rule provides a sufficient condition of the target class since it presents a quantitative measurement of the properties which occur in the target class versus that occurring in the contrasting classes. Therefore, the mined rule should be in the form of

$$\forall x \text{ target_class}(x) \leftarrow \text{condition}_1(x)[d : w_1] \vee \dots \vee \text{condition}_n(x)[d : w_n].$$

The rule indicates that if x satisfies condition_i , there is a possibility of w_i that x is in the *target_class*, where i is in $\{1, \dots, n\}$.

Based on the above discussion, the method for extraction of discriminant rules is outlined in the following algorithm.

Algorithm 5.1 (Discovery of discriminant rules in object-oriented databases) *Discovery of discriminant rules in object-oriented databases by dimension-based induction.*

- Input.**
1. A task for discovery of a discriminant rule which specifies a target class and one or a set of contrasting classes,
 2. $\text{Gen}(a_i)$, a set of concept hierarchies or generalization operators on attributes a_i , and
 3. T_i , a set of *attribute thresholds* for attributes a_i .

Output. A discriminant rule which distinguishes the major properties in the target class from those in the contrasting classes.

- Method.**
1. Collect the relevant set of data respectively into the target class and the contrasting classes by query processing.
 2. Extract the *prime target class* (the prime class corresponding to the initial working class in the target class) in a similar way as the dimension-based induction at mining characteristic rules (i.e., Algorithm 4.1). Then generalize the concepts of the initial working class(es) in the contrasting class(es) to the same level as those in the prime target class, which results in the *prime contrasting class(es)*.
 3. For generating qualitative discriminant rules, compare the tuples in the prime target class against those in the prime contrasting class(es), mark those tuples which overlap in both classes. The discriminating properties are represented by the unmarked tuples.
 4. For generating quantitative discriminant rules, compute the discriminating weight for each property and output the properties whose discriminating weight is close to 1 (together with the discriminating weight). \square

Rationale of Algorithm 5.1.

Step 1 collects the relevant set of data respectively into the target class and the contrasting classes and step 2 generalizes these classes synchronously to the same concept level and forms the prime target class and the prime contrasting class(es). Step 3 compares the tuples in the prime target class against those in the prime contrasting class(es) and marks those tuples which overlap in both classes. Since the unmarked tuples represents the properties which occur in the target class but not in the contrasting classes or vice versa, such tuples represent the discriminating properties among the classes. Thus the rules so generated are qualitative discriminant rules. Step 4 computes the discriminating weight for each property and outputs the properties whose discriminating weight is close to 1 (together with the discriminating weight), which quantitatively describes the discriminating behavior of each property. Thus the rules so generated are quantitative discriminant rules. \square

Example 5.3 In the same database as in Example 4.1, a discriminant rule can be discovered which compares and contrast the conditions of different sizes of houses.

Let the query for finding a discriminant rule be as follows.

```

mine discriminant rule
for 'BigHouse'
where P.Home.House_size = 'Big'
in contrast to 'SmallHouse'
where P.Home.House_size = 'Small'
from Person P
where P.Age >= 40 and P.Car.Maker = 'Japan'
      and 'Ph.D' in P.Education and P.Workplace.Name = 'U.B.C.'
with respect to P.Name, P.Home.House_size, P.Salary, P.Home.Address

```

The prime target class and the prime contrasting class, resulted from the generalization of the data in the target class and the contrasting class respectively, are shown in Table 5.

		<i>senior_professor</i>				<i>senior_manager</i>				<i>Total</i>
		<i>L</i>	<i>M</i>	<i>H</i>	<i>Total</i>	<i>L</i>	<i>M</i>	<i>H</i>	<i>Total</i>	
<i>Big</i>	<i>Richmond</i>	0	64	103	167	0	16	20	36	203
	<i>Burnaby</i>	2	27	46	75	1	8	9	18	93
	<i>Vancouver</i>	0	11	39	50	0	4	8	12	62
	<i>Total</i>	2	102	188	292	1	28	37	66	358
<i>Small</i>	<i>Richmond</i>	4	10	0	14	8	0	0	8	22
	<i>Burnaby</i>	11	4	2	17	2	9	0	11	28
	<i>Vancouver</i>	35	24	5	64	49	41	6	96	160
	<i>Total</i>	50	38	7	95	59	50	6	115	210

Table 5: A comparison of the situation related to big vs. small houses.

The following interesting rules can be generated from Table 5 *in comparison with the situations only related to big vs. small houses*.

```

if salary = 'low' then house_size = 'small' [97.32%]
if salary = 'high' and address = 'Richmond' then house_size = 'big' [100%]
if salary = 'low' and address= 'Vancouver' then house_size = 'small' [100%].

```

□

This method summarizes the major differences of a target class from a contrasting class at a high concept level. Notice that machine learning techniques, such as a decision-tree method like ID3 [48] and C4.5 [49] have been used to classify objects and find discriminating behaviors. To these algorithms, the dimension-based induction can be viewed as a preprocessing stage which performs a preliminary generalization using domain knowledge (such as concept hierarchies) to make rules expressible at a relatively high concept level to achieve both processing efficiency and representation clarity. Examples in the ID3 or C4.5 algorithms assume that concepts to be classified are already at a relatively high level, such as “mild” (temperature) and “high” (humidity) [48], which may not be the cases for detailed data stored in large databases but can be easily achieved by dimension-based generalization. After such generalization, one may choose either presenting directly the characteristic and/or discriminant rules as described above (which are in the relational form, desirable for some applications, and allowing the same tuples appearing in different classes but with weights associated), or integrating with an ID3-like algorithm for further concept classification and compact rule generation.

5.3 Extraction of classification rules with determinants

As mentioned above, after obtaining a prime generalized class, different techniques can be applied for further generalization, concept classification, attribute reduction, and compact rule generation. One may view that the derivation of a projected crosstab is in effect the reduction of the number of attributes in a generalized crosstab. The removal of some attributes in the further generalization of a prime class to derive the reduced generalized class(es) can also be viewed as a form of attribute reduction. Since there could be many relevant attributes in a data mining task, the selection of attributes to be reduced or further generalized is crucial to the interestingness of the generalized rules or generated crosstabs.

Previous studies on machine learning [39], statistics, fuzzy and rough set theories [57], etc. have developed many techniques at the evaluation of the importance of different sets of attributes and projection on the appropriate ones. A method which integrates the ID3 decision tree generation method [48, 49] and the dimension-based induction is presented here for the reduction of less informed attributes and the extraction of classification rules with determinants.

The method constructs a decision tree on top of the prime class, classifies the generalized objects in the prime class based on the class attributes, and leads to the generation of a set of classification rules.

In the construction of a decision tree, ID3 [48, 49] uses an information-theoretic approach which selects the attribute that provides the highest information gain to be the root of the (sub)-tree. This selection process minimizes the expected number of tests to classify an object and guarantees that a simple (but may not be the simplest) tree is found.

Let the prime class P contain p_i objects in class P_i (for $i = 1, \dots, m$), and each class P_i is distinguished from another class P_j ($i \neq j$) based on their different values in the class attribute. An arbitrary object shall belong to class P_i with probability p_i/p , where p is the total number of objects in the prime class P . When a decision tree is used to classify an object, it returns a class. A decision tree can thus be regarded as a source of messages for P_i 's with the expected information needed to generate this message given by

$$I(p_1, p_2, \dots, p_m) = - \sum_{i=1}^m \frac{p_i}{p} \log_2 \frac{p_i}{p}. \quad (5.6)$$

If an attribute A with values $\{a_1, a_2, \dots, a_k\}$ is used for the root of the decision tree, it will partition a class C into $\{C_1, C_2, \dots, C_k\}$, where C_j contains those objects in C that have value a_j of A . Let C_j contain p_{ij} objects of class P_i . The expected information required for the tree with A as the root is then obtained as the weighted average:

$$E(A) = \sum_{j=1}^k \frac{p_{1j} + \dots + p_{mj}}{p} I(p_1, \dots, p_m). \quad (5.7)$$

The information gained by branching on A is

$$gain(A) = I(p_1, p_2, \dots, p_m) - E(A). \quad (5.8)$$

ID3 examines all the candidate attributes and chooses an attribute A to maximize $gain(A)$ to form a tree. It then uses the same process recursively to form a decision tree for the residual subset C_1, C_2, \dots, C_m .

Notice that in the ID-3 method, the recursive classification process terminates when all the objects in a subclass belong to one determinant class. Because of the large amount and the wide diversity of data in databases, there is rarely the case that all the objects in one class belong to the same determinant class. Thus, a *classification threshold* should be set up such that further classification on a set of classified objects may become unnecessary if a substantial portion (i.e., no less than the prespecified classification threshold) of the classified objects belong to the same determinant class.

Based on the above discussion, the method for extraction of classification rules with determinant is outlined in the following algorithm.

Algorithm 5.2 (Discovery of classification rules with class attributes) *Discovery of classification rules with class attributes in object-oriented databases by integration of dimension-based induction with the ID3 method.*

- Input.**
1. A task for the discovery of classification rules of a specified set of data and a specified class attribute in an object-oriented database,
 2. $Gen(a_i)$, a set of concept hierarchies or generalization operators on attributes a_i ,
 3. T_i , a set of *attribute thresholds* for attributes a_i , and
 4. a classification threshold λ .

Output. A classification of the set of data and a set of classification rules.

- Method.**
1. Collect the relevant set of data by object-oriented query processing.
 2. Extract the *prime target class* (the prime class of the initial working class) in a similar way as the dimension-based induction at mining characteristic rules (i.e., Algorithm 4.1).

3. Compute the information gain for each candidate attribute based on an information-theoretic approach using the equations (5.6) – (5.8). Select one candidate attribute as the classifying attribute at this current level and classify the prime target class.
4. For each classified target subclass, repeat Step 3 to further classify it until either (1) all or a substantial proportion (no less than a prespecified classification threshold) of the objects are in one (determinant) class, or (2) no more classifying attributes can be used for further classification.
5. Generate rules according to the decision trees so derived. □

Rationale of Algorithm 5.2.

Steps 1 and 2 are verified in the generalization algorithm. Step 3 is essentially the ID3 algorithm whose correctness has been shown in [48, 49]. Step 4 determines whether a recursive, further classification need to be performed. If a substantial portion of the class of objects (no less than a classification threshold λ) belongs to one determinant class, the set of objects can be viewed as having been successfully classified and it is not necessary to perform further classification on it. Otherwise, further classification will be performed on the set of objects in a process similar to Step 3. The classification process terminates when either all the objects are so classified or no more classifying attributes can be used for further classification. Step 5 generates rules according to the decision tree so derived which reflects the general regularity of the data in the database. □

Example 5.4 The query in Example 5.1 can be minorly modified for discovery of a classification rule with one attribute specified as determinant, as shown below.

```
mine classification Generalized_P
according to class attribute Home.House_size
with classification threshold = 85%
```

The classification of the generalized relation, *Generalized_P*, is performed as follows.

1. Based on the extracted prime class table, Table 2, derived in Example 4.1, information gain is computed for each candidate attribute, *Occupation*, *Salary*, and *Address*, based on an information-theoretic approach and the equations (5.6) – (5.8).

<i>Attribute</i>	<i>Information Gain</i>
<i>Occupation</i>	0.0865259036
<i>Salary</i>	0.3586720686
<i>Address</i>	0.1888694432

Table 6: The information gain for each attribute to classify the prime class.

The computation results in Table 6, which implies that *Salary* should be chosen as the root of the decision tree, and the objects should be classified according to the values (*High*, *Medium*, *Low*) in the attribute “*Salary*”.

<i>Salary</i>	<i>House_size = Big</i>	<i>House_size = Medium</i>	<i>House_size = Small</i>
<i>High</i>	59.84%	36.70%	3.46%
<i>Medium</i>	41.93%	29.68%	28.39%
<i>Low</i>	2.63%	1.76%	95.61%

Table 7: Object distribution in each subclass.

The object distribution in each subclass is presented in Table 7. Since the determinant class, “*House_size = Small*”, contains 95.61% of objects in the subclass “*Salary: Low*”, which is above the classification threshold, it is unnecessary to further classify the subclass “*Salary: Low*”. Classification is further performed in other subclasses.

- For each remaining subclass to be further examined, repeat the first step to further classify it.

The computation of information gain for each remaining attribute in the two subclasses leads to Table 8. Obviously, *Address* should be chosen as the classifying attribute for both subclasses, “*Salary: High*” and “*Salary: Medium*”.

<i>Attribute</i>	<i>Information Gain</i>	
	<i>Salary: High</i>	<i>Salary: Medium</i>
<i>Occupation</i>	0.2164310451	0.3473828339
<i>Address</i>	0.2505954843	0.4922614419

Table 8: The information gain for each remaining attribute in the two subclasses.

	<i>Salary: High</i>			<i>Salary: Medium</i>		
	<i>Big</i>	<i>Medium</i>	<i>Small</i>	<i>Big</i>	<i>Medium</i>	<i>Small</i>
<i>Burnaby</i>	40.15%	58.39%	1.46%	48.61%	33.33%	18.06%
<i>Richmond</i>	85.42%	14.58%	0.00%	55.56%	37.50%	6.94%
<i>Vancouver</i>	49.47%	38.95%	11.58%	15.96%	14.89%	69.15%

Table 9: Object distribution in the subclass *Salary: High* and *Salary: Medium*

The object distribution in each subclass is presented in Table 9. Since the determinant class “*House_size = Big*” contains 85.42% of objects in the subclass “*Salary: High & Address: Richmond*”, which is above the classification threshold, it is unnecessary to further classify this subclass. Classification is further performed in other subclasses.

- For the remaining subclasses, based on the last classification attribute *Occupation*, the object distribution in each leaf class is presented in Tables 10 and 11, respectively.

	<i>Salary:High & Address:Burnaby</i>			<i>Salary:High & Address:Vancouver</i>		
	<i>Big</i>	<i>Medium</i>	<i>Small</i>	<i>Big</i>	<i>Medium</i>	<i>Small</i>
<i>s. professor</i>	41.82%	56.36%	1.82%	51.32%	42.10%	6.58%
<i>s. manager</i>	33.33%	66.67%	0.00%	42.10%	26.32%	31.58%

Table 10: Object distribution in the subclass *Salary: High* and *Salary: Medium*

- The classification leads to a decision tree of Figure 5.
- Classification rules can be generated with probability distribution information associated, based on the tables and the decision tree so derived. For example, the rules which associate house size with the salary can be derived as follows, by mapping Table 7 into the rule form.

if $Salary(x) = High$
then $(House_size(x) = Big)$ [59.84%] \vee $(House_size(x) = Medium)$ [36.70%]
 \vee $(House_size(x) = Small)$ [3.46%]
if $Salary(x) = Medium$
then $(House_size(x) = Big)$ [41.93%] \vee $(House_size(x) = Medium)$ [29.68%]
 \vee $(House_size(x) = Small)$ [28.39%]
if $Salary(x) = Low$
then $(House_size(x) = Big)$ [2.63%] \vee $(House_size(x) = Medium)$ [1.76%]
 \vee $(House_size(x) = Small)$ [95.61%]

	<i>Sal:Medium & Addr:Burnaby</i>			<i>Sal:Medium & Addr:Richmond</i>			<i>Sal:Medium & Addr:Vancouver</i>		
	<i>Big</i>	<i>Medium</i>	<i>Small</i>	<i>Big</i>	<i>Medium</i>	<i>Small</i>	<i>Big</i>	<i>Medium</i>	<i>Small</i>
<i>s. professor</i>	51.92%	40.38%	7.69%	55.17%	36.21%	8.62%	23.40%	25.53%	51.06%
<i>s. manager</i>	40.00%	15.00%	45.00%	57.14%	42.86%	0.00%	8.51%	4.26%	87.23%

Table 11: Object distribution in the subclass *Salary: High* and *Salary: Medium*

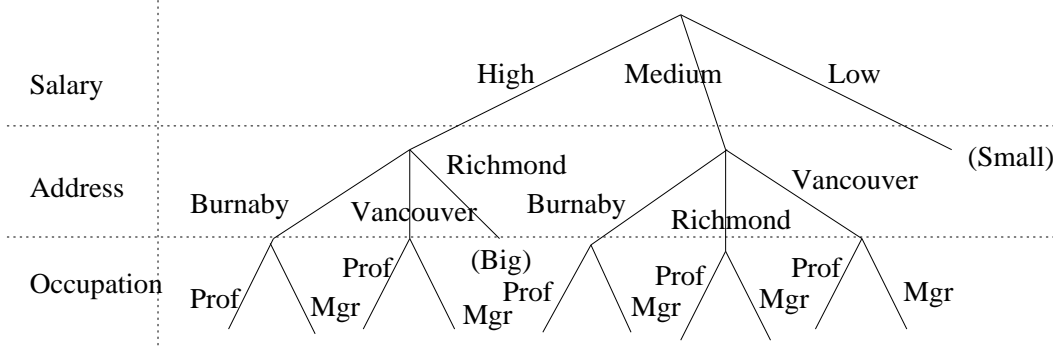


Figure 5: The decision tree generated based on the query and the class attribute *House_size*.

Rules associating *House_size* with *Salary* and *Address*, or with *Salary*, *Address*, and *Occupation*, can also be derived by a similar mapping from other tables. \square

Notice that alternative choices can be explored for determining whether a subclass need to be further classified. One such alternative is to specify a *noise threshold* and a *disjunct threshold* (the maximum number of disjuncts allowed). A subclass does not need to be further classified if it contains only a small number of disjuncts (below the disjunct threshold) after filtering each disjunct which is below the noise threshold. For example, in Table 7 of Example 5.4, if the noise threshold is 5% and the disjunct threshold is 2 (i.e., two conjuncts are allowed), subclasses, “*Salary : High*” and “*Salary : Low*”, contain only one or two disjuncts after filtering each disjunct below the noise threshold. Further classification needs to be performed only on one subclass, “*Salary : Medium*”.

5.4 Mining high-level association rules

An association rule is a rule in the form of

$$A_i \wedge \dots \wedge A_j \rightarrow B_k \wedge \dots \wedge B_l$$

where $A_i, \dots, A_j, B_k, \dots, B_l$ are sets of properties associated with the database.

An association rule can be expressed at a primitive concept level, i.e., every constant appearing in the rule also appears in the database, or be expressed at a generalized, high concept level, i.e., some or all the constants appearing in the rule are at a concept level higher than those stored in the database. For example, a rule

$$status(x, manager) \rightarrow house_size(x, big). \quad [0.08, 70\%]$$

is a high-level association rule because the constants in both status (= “*manager*”) and house_size (= “*big*”) are high-level constants.

Notice that the two numbers enclosed in the square brackets in the form of “[support, confidence]” are two important parameters to measure the significance of the rule [4]. The first parameter, **support** of a pattern p in a set S , denoted as $\sigma(p/S)$, is the probability that a pattern p appears in set S , i.e., the number of the tuples in S which contain p versus the total number of tuples in S . The second one, the **confidence** of $p \rightarrow q$ in S , $\varphi(p \rightarrow q/S)$, is the ratio of $\sigma(p \wedge q/S)$ versus $\sigma(p/S)$, i.e., the probability that a pattern q also occurs in S when the pattern p occurs in S .

Mining association rules in transaction databases has been studied extensively in recent database research, with several interesting algorithms developed for mining single- or multiple- level association rules in such databases [4,

35, 53, 27]. A similar association rule mining algorithm can be developed to adapt such rule mining technique in object-oriented databases. For example, following the study in [27], a multiple-level rule mining algorithm may first discover **large itemsets** (the set of items with a support no smaller than a prespecified support threshold) at a high concept level, and then examine their corresponding descendants.

The mining of multiple-level or primitive level association rules is a costly process because of the very large possibilities to group the large set of data items into different sized groups [4, 53, 27]. However, people may sometimes be only interested in the association relationships among the data at high concept levels. With the availability of the generalized prime class, an efficient mining technique can be developed for mining association rules at a concept level no lower than those expressed in the generalized prime class.

Here we outline the method for the extraction of high-level association rules from a generalized prime class.

A prime class preserves data relationships at a high concept level with *count* registered in each generalized tuple. Therefore, large data itemsets, i.e., the attribute-value pairs with the support no lower than the minimum support threshold σ , can be extracted from the prime class. A large 1-itemset is a single attribute-value pair with the support greater or equal to σ . Notice that the support of 1-itemset (i.e., single attribute-value pair) can be obtained by calculating the sum of the count for each attribute value in the prime class. The large 1-itemsets can be paired to form the candidate large 2-itemsets whose support can be computed by calculating the sum of the count for each of such 2-itemsets in the prime class. The large 2-itemsets are those candidates passing the support threshold testing. This process continues until the large k -itemsets are computed, where k is the maximum number of attributes in the prime class, or until no more large k -itemsets are derived for a certain value k . After the derivation of all the large itemsets, the association rules can be derived from these large itemsets after passing through the second filter: the minimum confidence threshold φ , based on the definition of the minimum confidence threshold.

This process is summarized in the following algorithm.

Algorithm 5.3 (Discovery of high-level association rules in object-oriented databases) *Discovery of high-level association rules in object-oriented databases using the prime (generalized) class.*

- Input.**
1. A task for the discovery of association rules from a specified set of data at a high concept level in an object-oriented database,
 2. $Gen(a_i)$, a set of concept hierarchies or generalization operators on attributes a_i ,
 3. T_i , a set of *attribute thresholds* for attributes a_i , and
 4. a support threshold σ and a confidence threshold φ .

Output. A set of association rules for the specified set of data at high concept levels.

- Method.**
1. Collect the relevant set of data (i.e., the initial working class) by object-oriented query processing.
 2. Extract the *prime class* of the initial working class in a similar way as the attribute-oriented induction at learning characteristic rules (i.e., Algorithm 4.1).
 3. Compute the large k -itemsets based on the data in the *prime class*, which is performed as follows.
 - (a) Compute the large 1-itemsets L_1 as follows.
Compute the count of each 1-item " $A_1 = a_1$ " by summing up all the counts of the tuples in the prime class with " $A_1 = a_1$ ". Collect into L_1 all the 1-items whose support passes the support threshold test, i.e., if $(A_1 = a_1).count \geq \sigma \times ||DB||$, then $L_1 = L_1 \cup (A_1 = a_1)$.
 - (b) Iteratively compute the large k -itemsets L_k (for $k > 1$) based on the idea of the Apriori algorithm [4]. The k -th iteration is described as follows.
Generate the candidate large k -itemsets by Apriori: first forming a k -item using two neighboring $(k-1)$ -items in L_{k-1} and then testing whether all of its $(k-1)$ -item subset is in L_{k-1} . Only if the test is passed, the k -item is enclosed in the candidate large k -itemset C_k .
Compute the support of each large k -item based on the count in the prime class, and insert it into L_k only if its support is no less than the support threshold σ .
 - (c) The process repeats until all the large k -itemsets are computed, where k is the maximum number of attributes in the prime class, or until no more large k -itemsets are derived for a certain value k .

4. Perform the confidence testing and output the set of association rules which pass the test. Each association rule is in the form of

$$A_i \wedge \cdots \wedge A_j \rightarrow B_m \wedge \cdots \wedge B_n$$

where $A_i, \dots, A_j, B_m, \dots, B_n$ are the attribute-value pairs of a large k -itemset (for $k = j - i + n - m + 2$) collected at Step 3. \square

Rationale of Algorithm 5.3.

Steps 1 and 2 are verified in the generalization algorithm. Step 3 is essentially the Apriori association rule computation algorithm whose correctness has been shown in [4]. Notice the major difference of the computation at this step is that the support (i.e., count) of a pattern is computed using the data not in the initial working class but in the prime class. This involves much less database accesses than computing at the primitive concept level. Step 4 generates all the rules after passing the confidence testing, which is similar to [4]. Therefore, all the association rules at the concept level of the prime class is generated, which reflects the association relationships at the prime class concept level for the relevant set of data in the database. \square

The following example illustrates the association rule derivation process.

Example 5.5 In the same database as in Example 4.1, one may discover a set of high-level association rule which reflect the relationships among different attributes at the concept levels represented in the prime class.

Let the query for finding association rules be as follows, with the minimum support threshold being 0.15 and the minimum confidence threshold 0.7.

```
mine association rules
from Person P
where P.Age >= 40 and P.Car.Maker = 'Japan'
      and 'Ph.D' in P.Education and P.Workplace.Name = 'U.B.C.'
with respect to P.Name, P.Home.House_size, P.Salary, P.Home.Address
```

The derivation of the association rules proceeds as follows. First, following Algorithm 5.3, Step 1 collect the relevant set of data, and Step 2 derives the prime class Table 1. Both are the same as Example 4.1.

At Step 3, the large 1-itemset L_1 is first discovered from Table 1, which is shown in Table 12.

large 1-itemset	support
<i>class</i> = "senior_professor"	72.12%
<i>class</i> = "senior_manager"	27.88%
<i>house_size</i> = "big"	44.75%
<i>house_size</i> = "medium"	29.00%
<i>house_size</i> = "small"	26.25%
<i>address</i> = "richmond"	37.62%
<i>address</i> = "burnaby"	28.12%
<i>address</i> = "vancouver"	34.25%
<i>salary</i> = "medium"	38.75%
<i>salary</i> = "high"	47.00%

Table 12: The large 1-itemsets in the prime class.

Pairing these large 1-items forms the candidate large 2-itemsets whose support can be calculated using the prime class. This leads to the large 2-itemsets L_2 as shown in Table 13.

Pair the large 2-itemsets using the Apriori algorithm to form candidate large 3-itemsets. Calculate their supports using the prime class and check them against the support threshold. Thus, the large 3-itemsets L_3 is obtained, as shown in Table 14.

Similarly, the candidate large 4-itemsets are formed. Since there is no large 4-itemsets L_4 discovered with sufficient support. The iterative derivation at Step 3 terminates.

At step 4, the confidence threshold test outputs the final set of associations rules, as shown in Table 15, which is the answer to the query. \square

large 2-itemset	support
{class = "senior_professor", house_size = "big"}	36.50%
{class = "senior_professor", address = "richmond"}	30.50%
{class = "senior_professor", salary = "high"}	38.62%
{class = "senior_professor", house_size = "medium"}	23.75%
{class = "senior_professor", address = "burnaby"}	21.88%
{class = "senior_professor", salary = "medium"}	26.88%
{class = "senior_professor", address = "vancouver"}	19.75%
{house_size = "big", address = "richmond"}	25.37%
{house_size = "big", salary = "high"}	28.12%
{house_size = "big", salary = "medium"}	16.25%
{house_size = "medium", salary = "high"}	17.25%
{house_size = "small", address = "vancouver"}	20.00%
{address = "richmond", salary = "high"}	18.00%
{address = "richmond", salary = "medium"}	18.00%
{address = "burnaby", salary = "high"}	17.12%

Table 13: The large 2-itemsets in the prime class.

large 3-itemset	support
{class = "senior_professor", house_size = "big", address = "richmond"}	20.88%
{class = "senior_professor", house_size = "big", salary = "high"}	23.50%
{class = "senior_professor", address = "richmond", salary = "high"}	15.38%
{house_size = "big", address = "richmond", salary = "high"}	15.38%

Table 14: The large 3-itemsets in the prime class.

6 Discussion

Previous three sections presented techniques for object and class generalization and methods for generalization-based discovery of several kinds of knowledge in object-oriented databases, including characteristic rules, discriminant rules, association rules, and object classification with class attributes.

In this section, we discuss several related issues, including the philosophy of generalization-based knowledge discovery in object-oriented databases, further development of knowledge discovery mechanisms in object-oriented databases, and the application of discovered knowledge.

6.1 Philosophy of generalization-based knowledge discovery in object-oriented databases

This study is focused on a generalization-based knowledge discovery mechanism for data mining in object-oriented databases. The general philosophy of the mechanism is based on the fact that a database usually contains a huge amount of complex data and objects, and database users may like to view any relevant set of this data at a generalized, high concept level in a flexible way from different angles. The generalization of a large set of data into a relatively small set of generalized data associated with corresponding statistical information facilitates this process. By generalization of data to a relatively high level, different kinds of feature tables, generalized profiles, and other kinds of generalized knowledge can be extracted and presented with minimal overhead, as demonstrated in Section 5.

The class-based generalization method proposed here adopts an dimension-based generalization technique. A major strength of dimension-based induction over tuple-oriented ones is at its efficiency in the induction process. The former performs generalization on each attribute *uniformly* for all the tuples in the initial class in the formation of the prime class, which contrasts with the latter (i.e., the tuple-oriented induction) that explores different possible combinations for a large number of tuples in the same generalization phase. An exploration of different possible generalization paths

$house_size = "big" \rightarrow class = "senior_professor"$	[36.50%, 81.56%]
$house_size = "medium" \rightarrow class = "senior_professor"$	[23.75%, 81.90%]
$house_size = "small" \rightarrow address = "vancouver"$	[20.00%, 76.19%]
$address = "richmond" \rightarrow class = "senior_professor"$	[30.50%, 81.06%]
$address = "burnaby" \rightarrow class = "senior_professor"$	[21.88%, 77.78%]
$salary = "high" \rightarrow class = "senior_professor"$	[38.62%, 82.18%]
$house_size = "big" \wedge address = "richmond" \rightarrow class = "senior_professor"$	[20.88%, 82.27%]
$house_size = "big" \wedge salary = "high" \rightarrow class = "senior_professor"$	[23.50%, 83.56%]
$address = "richmond" \wedge salary = "high" \rightarrow class = "senior_professor"$	[15.38%, 85.42%]
$address = "richmond" \wedge salary = "high" \rightarrow house_size = "big"$	[15.38%, 85.42%]

Table 15: The high-level association rules generated.

for different tuples at the early generalization stage will not be productive since these combinations will be merged in further generalization. Different possible combinations should be explored only when the class has been generalized to a relatively small prime class.

Dimension-based induction is robust and handles noise and/or exceptional cases elegantly because it incorporates statistical information (using *count* and other aggregate functions, such as *sum*, *total*, *max*, *min*, *average*) and generates disjunctive rules. The association of *count* with each disjunct leads naturally to mining *approximate rules* for which the conditions with negligible weight can be dropped in generalization and rule formation since a negligible weight implies a minimal influence to the conclusion.

Count association facilitates incremental learning in large databases: when a new tuple is inserted into a database, its concepts (attribute values) are first generalized to the same concept level as those in the generalized class and then merged naturally (by count incremental) into the generalized class if there exists one such generalized tuple, or otherwise inserting a new generalized tuple into the generalized class.

Furthermore, with the association of count information, data sampling [34] and parallelism can be explored in knowledge discovery. Dimension-based induction can be performed by sampling a subset of data from a huge set of relevant data or by first performing induction in parallel on several partitions of the relevant data set and then merging the generalized results.

As a generalization-based method, dimension-based induction confines its power to the discovery of knowledge rules at general concept levels. There are applications which require the discovery of knowledge at primitive concept levels, such as finding (primitive) association or dependency rules, or finding functional relationships (e.g., " $y = f(x)$ ") between two numerical attributes (e.g., height and weight). Dimension-based induction does not suit such applications. Moreover, dimension-based induction needs reasonably informative concept hierarchies or generalization operators for generalization of nonnumerical attributes, and over-generalization should be guarded against by setting thresholds flexibly and/or interactively. Nevertheless, the method is useful in most database-oriented applications which need to generalize some or all of the relevant attributes.

6.2 Discovery of other kinds of knowledges

Besides the extraction of the kinds of rules presented in Section 5 from object-oriented data, class generalization-based knowledge discovery techniques can also be used for mining other kinds of knowledge, including data evolution regularities, data deviation rules, and data clustering. Methods for mining these kinds of rules are briefly outlined in this subsection.

A *data evolution rule* [53, 2] reflects the general evolution (or changing) behavior of a set of data, e.g., the rule which describes the major factors which influence the fluctuations of certain stock values. The discovery of data evolution regularities is to mine the general characteristics of the set of data changing with time. Such characteristics may include the description of the general behavior of a set of data fluctuating with time, comparison of the behaviors between different sets of data changing with time, classification of data based on their time-sensitive behaviors, association of a set of properties related to time, etc.

Obviously, the methods discussed in previous sections for mining different kinds of rules can be extended to temporal or other time-related data as well. One distinct feature for time-related data is that time can be viewed as a special and

distinct dimension (attribute) in data mining. Time can often be arranged into concept hierarchies or lattices based on the commonly accepted hierarchy of time. For example, *second, minute, hour, day, month, quarter, year, decade, century* can be viewed as a concept hierarchy. However, if some other time slots such as *week* and *semester* need to be incorporated into the time “hierarchy”, the “hierarchy” may become a lattice since weeks or semesters usually cross the boundaries of month, quarter, etc. These predefined “time hierarchies” (provided in most commercial database systems) are valuable knowledge and are useful in data generalization and mining interesting rules.

A **data deviation rule** summarizes the behavior of a set of data which deviates (substantially or to certain degree) from the general trend or behavior of the majority set of data in the database. For example, one may observe the sales history of certain commodities in the market and find that there are some products which may have far better or far worse sales than the majority ones. The task for mining data deviation regularities is to discover such sets of data and characterize their general behaviors.

Methods for mining data deviation regularities may include two subtasks: (1) finding the general trend or behavior of data in the database, to which the methods discussed in the previous sections and paragraphs may apply; and (2) mining the deviation data and characterizing their behaviors. To perform the second task, similarity measurements should be defined and applied [2], which may specify to what extent the different data may still be considered as similar but to what extent it should be categorized as deviations. Fuzzy logics or uncertainty measurements may be associated with the deviated data to quantitatively characterize the behaviors of the data deviations. Until recently, we have not found many studies on database-oriented algorithms for mining data deviation regularities. More studies need to be performed in this direction.

A **data clustering** task is to cluster a set of objects in object-oriented databases based on the similarity of objects according to certain criterion. A commonly accepted object clustering criterion is the principle of conceptual clustering [18]: *clustering a set of objects in an attempt to maximize intraclass similarity and inter-class differences*.

Data clustering has been studied in statistics [11], machine learning [13, 18, 19], and databases [42, 15] with different methods and different emphases. Previous approaches, probability-based (like most approaches in machine learning) or distance-based (like many methods in statistics), do not adequately consider the cases the data sets can be too large to fit into main memory. Recent studies on data mining by clustering techniques [42, 15] have addressed this problem, and some relatively efficient methods, such as CLARANS [42], R*-tree-based classification [15], and BIRCH algorithm [55] seem to be in the promising direction. Based on our observation, similar techniques can be applied to object clustering in object-oriented databases as well. However, the algorithm development and performance study of the possible extensions of the methods toward object-oriented databases forms a challenging task which is left to the future study.

6.3 Application of discovered knowledge

Data mining in object-oriented databases broadens substantially the spectrum of applications of object-oriented database technology. Besides finding some interesting, different kinds of knowledge for database users and administrators, mining object-oriented data may provide a multiple dimensional view on any subset of object-oriented data, promote schema modification and evolution based on database contents, perform intelligent query answering using data mining tools and discovered knowledge, and facilitate query optimization using discovered knowledge. These new applications are briefly discussed in this subsection.

6.3.1 A multiple dimensional view of a set of object-oriented data

The generalization of a set of data in an object-oriented database into a generalized class (such as the *prime class*) can be viewed as forming an n -dimensional cube, where n is the number of attributes in the generalized class. Each dimension of the cube corresponds to one attribute, consisting of a set of (generalized) attribute-values, and each cube cell contains the count and/or other aggregate values (such as sum, average, max, min, etc.), representing the statistical information or aggregated data of the corresponding cell.

This n -dimensional base cuboid can be further generalized in many ways to form a set of smaller, but more generalized cuboids, which together with the original n -dimensional cuboid, form the lattice of a multi-dimensional cube. The formation of such a multi-dimensional data cube may facilitate data mining or on-line analytical processing (OLAP), which are useful in business management and decision making, as shown in many studies, such as [30, 23].

Let the lowest level n -dimensional base cuboid stored in a database be C_0 , which is obtained by the dimension-based generalization technique illustrated in Section 4. Let a sequence of multi-dimensional cuboids generalized from it be

C_1, C_2, \dots, C_k , whose relationships form a lattice. Starting from a multi-dimensional cuboid C_i , one can (1) “roll-up” the cube by either going up the lattice to a corresponding higher concept leveled cuboid, or, when such a node is not available in the lattice, performing further generalization as illustrated in Section 5.1; or (2) “drill-down” the cube by either going down the lattice to a corresponding lower concept leveled cuboid, or, when such a node is not available in the lattice, performing generalization to appropriate levels from a lower level cuboid, or if there is none existent, from the base cuboid C_0 , as illustrated in Sections 4 and 5.1.

Obviously, the dimension-based generalization may play an important role in the formation of such a lattice of multi-dimensional cuboids or implementing the object cube transformation/traversal operations, such as “roll-up” and “drill-down”.

6.3.2 Schema modification and evolution based on database contents

Previous studies on schema modification and evolution are mainly based on database integrity constraints. With the methods developed for mining data at a high concept level and classification of generalized data, one may explore schema modification and evolution based on database contents. Obviously, schema modification and evolution based on database contents and data distributions at a high concept level is a highly desirable approach since it is based on an objective measurement, data, rather than on subjective measurement, some expert knowledge, which are often incomplete, biased and error-prone.

A simple approach for schema modification could be providing a set of multiple-level cuboids from different angles and let users interact with the data mining system to see which view provides clean data semantics and desirable data distribution. For example, in a university registration database, if the students’ registration are partitioned nicely according to the department, it is clean to partition the student objects (to form class/subclass hierarchies) according to the departments they are in. However, if the student (such as freshman) registered courses cannot be partitioned nicely based on the department, it could be partitioned according to other criteria, such as number of years in the program, etc. Such partitioning can be obtained by interaction between a data mining system and a database administrator.

6.3.3 Intelligent query answering using data mining tools and discovered knowledge

The regularities discovered by generalization in OODBs are also useful in both querying data and knowledge stored implicitly or explicitly in OODBs and answering queries intelligently.

Example 6.1 Let the database and conceptual hierarchies be the same as that presented in Example 4.1. We examine how conceptual hierarchies and knowledge discovery tools may help answer queries intelligently.

First, generalization may substantially increase the power and flexibility of querying OODBs. For example, to find everyone who has graduate degree, works in U.B.C., living in a big and expensive house in Burnaby. One may formulate a query containing high level concepts, such as, $P.home.house_size = \text{“big”}$, $P.home.house_value = \text{“expensive”}$, $graduate_degree$ in $P.education$, etc. Such concepts cannot be expressed in a query without the help of concept generalization since database stores only the floor-area or the price of a house but not high-level concepts like “big”, or “expensive”. With the availability of concept hierarchies and generalization operators, the concrete data in the database, such as the house value can be generalized into “expensive”, “cheap”, etc. by applying a sequence of *Gen* operations.

With the availability of generalization tools, queries can be answered in a general and interesting manner [41]. Suppose the learning request of Example 4.1 is rewritten as a query. That is, the request is to “**select** P.Name, ... **from** Person P **where** ...” which is to find the names, etc. of all the persons who satisfy the same condition as the learning request in Example 4.1. A strict query answering will print the names and other inquired information for every 599 persons, which could be very boring to users. An intelligent way to answer the query is to summarize the information and print the general characteristics as demonstrated by Example 4.1. The detailed information for 599 persons will be printed only when the user is not satisfied with the general answers. Obviously, answers expressed at high concept levels are more attractive to most users than those at primitive levels. □

6.3.4 Query optimization using discovered knowledge

With the availability of discovered knowledge, general knowledge about database contents are made explicit, which may help improve the performance of query and transaction processing in object-oriented databases.

For example, if a data miner discovers that all the students who take a certain computer course are *graduate* students. The processing of a query related to the students taking this course can be confined to the search in the class of graduate students only, if the discovered knowledge is saved to facilitate the search.

Such a query optimization process is similar to semantic query optimization which applies integrity constraints to optimize a query. However, there are two major differences of query optimization using discovered knowledge from that of integrity constraints. First, the integrity constraint is a semantic constraint at the database (schema) definition level, i.e., it defines what is a valid database; however, the discovered knowledge is a constraint at the database content level, which reflects only the current database status, i.e., any update may revise or invalidate the discovered knowledge. Therefore, it is subtle to maintain such discovered knowledge for query optimization. Second, the discovered knowledge usually contains some uncertainty, such as “*80% students have excellent GPA in this department*”. Applying such knowledge may not exclude the search of other object classes if the query is to find all the answers. Therefore, it may facilitate optimization of queries for finding some of the answers or answering queries in an approximate or intelligent manners.

It is necessary to perform further study of query optimization using discovered knowledge in both relational and object-oriented databases.

7 Conclusions

A relatively systematic study about the issues and techniques for knowledge discovery in object-oriented databases has been conducted here, with the following issues examined.

1. Philosophical considerations and general methodologies for knowledge discovery in object-oriented databases are discussed.
2. Sophisticated generalization operators are examined for generalizing and handling complex data objects, such as structured data, methods, inherited data, object identifiers, class/subclass hierarchies, multimedia data, etc.
3. Generalization mechanisms are developed for object cube construction using a dimension-based class generalization mechanism. With the development of generalization operators and generalization control mechanisms, knowledge discovery can be performed in OODBs efficiently and effectively, which has been shown by the complexity analysis of the developed techniques.
4. Different kinds of knowledge, including characteristic rules, discriminant rules, object classification, and high-level association rules, can be extracted effectively from the generalized data. The interactive mining of multiple kinds of knowledge may substantially enhance the power and the flexibility of knowledge discovery in OODBs.
5. The philosophy of generalization-based data mining, the mining of other kinds of knowledge, and the application of discovered knowledge are also discussed in the study.

The techniques developed in this study are centered around a set-oriented, generalization-based data mining method, dimension-based induction. The method extracts effectively and efficiently different kinds of knowledge rules, and is also robust, extensible, and with wide applications. However, the method is a generalization-based technique and requires the availability of some background knowledge (such as concept hierarchies). Thus it may not be suitable for mining knowledge without generalization or without background knowledge.

Knowledge discovery in object-oriented database shares many similarities with knowledge discovery in relational database, especially when object-oriented data are generalized into prime generalized classes, which are often in the form of generalized tables or cuboids. However, methods for generalization of complex structured and unstructured data, class/subclass hierarchies, inherited and derived data, methods, and class composition hierarchies, form a new frontier and a broad spectrum for knowledge discovery in object-oriented databases.

There are many research issues on knowledge discovery in OODBs. The construction of efficient and effective generalization operators for complex structured or unstructured data, especially hypertext and multimedia data, is an important issue for further study. The construction of a multi-resolution model for OODBs [50], which may help browse OODB contents and answer interesting queries at high concept levels is another interesting topic for further study. Furthermore, integration of dimension-based induction with other data mining methods [1, 14, 34, 40, 47, 52, 54, 57] for mining various kinds of knowledge from object-oriented databases is another important direction in the future. Finally, software development and experimentation should be performed on the proposed mechanisms for data mining

in OODBs to verify and improve the proposed technique and compare it with other related, promising proposals [47, 45, 36].

In general, data mining provides a powerful tool for automatic generation and verification of knowledge in the construction of large knowledge-bases. It represents an important and promising direction in the development of data and knowledge-base systems. Our preliminary study of the data mining methods led to an efficient implementation of a data mining system, *DBMiner* [25], for relational databases. The construction of such a data miner for knowledge discovery in object-oriented databases is being performed in Osaka University in Japan, based on the methods studied in this paper. Preliminary experiments with the system prototype have shown that interesting knowledge can be discovered from object-oriented databases. Further experiments and performance studies with this object-oriented data mining system prototype will be reported in the future.

References

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., May 1993.
- [2] R. Agrawal, K.-I. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 490–501, Zurich, Switzerland, Sept. 1995.
- [3] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger. The Quest data mining system. In *Proc. 1996 Int'l Conf. on Data Mining and Knowledge Discovery (KDD'96)*, pages 244–249, Portland, Oregon, August 1996.
- [4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.
- [5] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database systems manifesto. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 223–240. Elsevier Science, 1990.
- [6] F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System: The story of O2*. Morgan Kaufmann, 1992.
- [7] E. Bertino and L. Martino. *Object-Oriented Database Systems*. Addison-Wesley, 1994.
- [8] R. J. Brachman. Viewing data from a knowledge representation lens. In *Proc. Int. Conf. Building and Sharing of Very Large-Scale Knowledge Bases'93*, pages 117–120, Tokyo, Japan, December 1993.
- [9] C. Brunk, J. Kelly, and R. Kohavi. MineSet: An integrated system for data mining. In *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pages 135–138, Newport Beach, California, August 1997.
- [10] R.G.G. Cattell. *Object Data Management: Object-Oriented and Extended Relational Databases, Rev. Ed.* Addison-Wesley, 1994.
- [11] K. C. C. Chan and A. K. C. Wong. A statistical technique for extracting classificatory knowledge from databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 107–124. AAAI/MIT Press, 1991.
- [12] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.
- [13] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: a bayesian classification system. In *Proc. Fifth Int. Conf. on Machine Learning*, pages 54–64, San Mateo, California, 1988.
- [14] W. W. Chu and K. Chiang. Abstraction of high level concepts from numerical values in databases. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 133–144, Seattle, WA, July 1994.
- [15] M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. In *Proc. 4th Int. Symp. on Large Spatial Databases (SSD'95)*, pages 67–82, Portland, Maine, August 1995.
- [16] U. M. Fayyad, S. G. Djorgovski, and N. Weir. Automating the analysis and cataloging of sky surveys. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 471–493. AAAI/MIT Press, 1996.
- [17] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
- [18] D. Fisher. Improving inference through conceptual clustering. In *Proc. 1987 AAAI Conf.*, pages 461–465, Seattle, Washington, July 1987.

- [19] D. Fisher. Optimization and simplification of hierarchical clusterings. In *Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining (KDD'95)*, pages 118–123, Montreal, Canada, Aug. 1995.
- [20] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1–27. AAAI/MIT Press, 1991.
- [21] B. R. Gaines and J. H. Boose. *Knowledge Acquisition for Knowledge-Based Systems*. London: Academic, 1988.
- [22] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.
- [23] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–54, 1997.
- [24] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.
- [25] J. Han, J. Chiang, S. Chee, J. Chen, Q. Chen, S. Cheng, W. Gong, M. Kamber, G. Liu, K. Koperski, Y. Lu, N. Stefanovic, L. Winstone, B. Xia, O. R. Zaiane, S. Zhang, and H. Zhu. DBMiner: A system for data mining in relational databases and data warehouses. In *Proc. CASCON'97*, Toronto, Canada, November 1997.
- [26] J. Han and Y. Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 157–168, Seattle, WA, July 1994.
- [27] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 420–431, Zurich, Switzerland, Sept. 1995.
- [28] J. Han and Y. Fu. Exploration of the power of attribute-oriented induction in data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 399–421. AAAI/MIT Press, 1996.
- [29] J. Han, S. Nishio, and H. Kawano. Knowledge discovery in object-oriented and active databases. In F. Fuchi and T. Yokoi, editors, *Knowledge Building and Knowledge Sharing*, pages 221–230. Ohmsha, Ltd. and IOS Press, 1994.
- [30] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 205–216, Montreal, Canada, June 1996.
- [31] K. Higa, M. Morrison, J. Morrison, and O. R. Sheng. An object-oriented methodology for knowledge base/database coupling. *Comm. ACM*, 35:99–113, June 1992.
- [32] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented database. In *Proc. 1992 ACM-SIGMOD Int. Conf. Management of Data*, pages 393–402, San Diego, CA, June 1992.
- [33] W. Kim. *Introduction to Object-Oriented Databases*. The MIT Press, 1990.
- [34] J. Kivinen and H. Mannila. The power of sampling in knowledge discovery. In *Proc. 13th ACM Symp. Principles of Database Systems*, pages 77–85, Minneapolis, MN, May 1994.
- [35] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int. Conf. on Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, Nov. 1994.
- [36] W. Klösgen. Explora: a multipattern and multistrategy discovery assistant. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 249–271. AAAI/MIT Press, 1996.
- [37] M. Manago and Y. Kodratoff. Induction of decision trees from complex structured data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 289–306. AAAI/MIT Press, 1991.
- [38] R. S. Michalski. A theory and methodology of inductive learning. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 83–134. Morgan Kaufmann, 1983.
- [39] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 2*. Morgan Kaufmann, 1986.
- [40] R. S. Michalski, L. Kerschberg, K. A. Kaufman, and J.S. Ribeiro. Mining for knowledge in databases: The INLEN architecture, initial implementation and first results. *J. Int. Info. Systems*, 1:85–114, 1992.
- [41] A. Motro and Q. Yuan. Querying database knowledge. In *Proc. 1990 ACM-SIGMOD Int. Conf. Management of Data*, pages 173–183, Atlantic City, NJ, June 1990.
- [42] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 144–155, Santiago, Chile, September 1994.

- [43] S. Nishio, H. Kawano, and J. Han. Knowledge discovery in object-oriented databases: The first step. In *Proc. AAAI-93 Workshop on knowledge discovery in databases*, pages 186–198, Washington, DC, July 1993.
- [44] J. Orenstein, S. Haradhvala, B. Margulie, and D. Sakahara. Query processing in the objectstore database system. In *Proc. 1992 ACM-SIGMOD Int. Conf. Management of Data*, pages 403–412, San Diego, CA, June 1992.
- [45] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–238. AAAI/MIT Press, 1991.
- [46] G. Piatetsky-Shapiro, U. Fayyad, and P. Smith. From data mining to knowledge discovery: An overview. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–35. AAAI/MIT Press, 1996.
- [47] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.
- [48] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [49] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [50] R.L. Read, D.S. Fussell, and A. Silberschatz. A multi-resolution relational data model. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 139–150, Vancouver, Canada, Aug. 1992.
- [51] G.M. Shaw and S. B. Zdonik. A query algebra for object-oriented databases. In *Proc. 6th Int. Conf. Data Engineering*, pages 154–162, Los Angeles, CA, February 1990.
- [52] W. Shen, B. Mitbander, K. Ong, and C. Zaniolo. Using metaqueries to integrate inductive learning and deductive database technology. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 335–346, Seattle, WA, July 1994.
- [53] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 407–419, Zurich, Switzerland, Sept. 1995.
- [54] R. Uthurusamy, U. M. Fayyad, and S. Spnggler. Learning useful rules from inconclusive data. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 141–157. AAAI/MIT Press, 1991.
- [55] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 103–114, Montreal, Canada, June 1996.
- [56] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 159–170, Tucson, Arizona, May 1997.
- [57] W. Ziarko. *Rough Sets, Fuzzy Sets and Knowledge Discovery*. Springer-Verlag, 1994.
- [58] J. Zytkow and J. Baker. Interactive mining of regularities in databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 31–54. AAAI/MIT Press, 1991.